
175 PTAS
MI COMPUTER
55
CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR
I, ROBOT
ASIMOV
I, ROBOT
ASIMOV
Harry Harrison

MI COMPUTER CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona

Volumen V - Fascículo 55

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones, Francisco Martín
Asesor técnico: Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, F. Martín, S. Tarditti, A. Cuevas, F. Blasco
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Paseo de Gracia, 88, 5.°, 08008 Barcelona
Tels. (93) 215 10 32 / (93) 215 10 50 - Télex 97848 EDLTE

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 96 fascículos de aparición semanal, encuadernables en ocho volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.

© 1983 Orbis Publishing Ltd., London
© 1984 Editorial Delta, S.A., Barcelona
ISBN: 84-85822-83-8 (fascículo) 84-7598-007-4 (tomo 5)
84-85822-82-X (obra completa)
Depósito Legal: B. 52/1984

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 308501
Impreso en España - Printed in Spain - Enero 1985

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, 28034 Madrid.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Edificio Bloque Dearmas, final Avda. San Martín con final Avda. La Paz, Caracas 1010.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

Servicio de suscripciones y atrasados (sólo para España)

Las condiciones de suscripción a la obra completa (96 fascículos más las tapas, guardas y transferibles para la confección de los 8 volúmenes) son las siguientes:

a) Un pago único anticipado de 19 425 ptas. o bien 8 pagos trimestrales anticipados y consecutivos de 2 429 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 6.850.277 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Editorial Delta, S.A. (Paseo de Gracia, 88, 5.°, 08008 Barcelona), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Editorial Delta, S.A., en la forma establecida en el apartado b).

Para cualquier aclaración, telefonear al (93) 215 75 21.

No se efectúan envíos contra reembolso.





Seres mecánicos

Iniciamos una serie en la que estudiaremos la ciencia de la robótica. En primer lugar, haremos un recuento histórico

Durante siglos muchas personas se han sentido atraídas, de una u otra forma, por la idea y la realización práctica de "hombres mecánicos". Filósofos, ingenieros e inventores se han abocado a la creación de máquinas que imiten el comportamiento humano. A pesar de que en la actualidad los robots se asemejan un poco a los seres humanos y se diseñan para realizar una gama de acciones específicas, los primeros hombres mecánicos se configuraron para que se parecieran en la mayor medida posible al hombre y sugirieran que podían realizar cualquier acción humana.

Al primer robot mecánico, sin embargo, no se le dio forma de hombre. En 1738, el ingeniero francés Jacques de Vaucanson (1709-1782) presentó un pato mecánico ante la Académie Royale des Sciences de París. El pato podía agitar las alas, graznar e ingerir alimentos. Luego, durante el mismo siglo XVIII, un inventor suizo, Pierre Jacquet-Droz (1721-1790), creó un juego de muñecos mecánicos que podían realizar diversas acciones. Uno escribía, otro dibujaba y un tercero tocaba música en un órgano. A fines del siglo XIX ya existía gran número de tales máquinas, todas basadas en mecanismos de relojería.

En Gran Bretaña, durante la época victoriana, se construyeron numerosas figuras de aspecto notablemente natural, y no todas eran de relojería. En 1893 George Moore creó un hombre mecánico cuya capacidad de movimientos se la proporcionaba la energía a vapor; un interesante efecto colateral del mismo consistía en hacer que el hombre mecánico aspirara un cigarro y pareciera exhalar el humo.

Las tecnologías más recientes han estimulado el desarrollo de máquinas más ambiciosas: desde los sencillos hombres mecánicos construidos con piezas del juego Meccano, capaces de caminar, hasta el clásico "Elektro", construido por la empresa norteamericana Westinghouse. El "Elektro" era un hombre mecánico de 2,15 m de altura que podía articular hasta 80 palabras diferentes, contar, caminar, hablar, saludar y distinguir entre distintos colores. Estaba alimentado por no menos de 11 motores eléctricos y pesaba 117 kg. Un "cerebro" compuesto por un total de 82 relés diferentes controlaba esta enorme mole.

Pero cada uno de estos hombres mecánicos tenía sus limitaciones. Ninguno de ellos, a pesar de su evidente valor como fuente de entretenimiento, poseía alguna de las habilidades que uno desearía hallar cuando piensa en el robot ideal. Un hombre mecánico que sepa dibujar no puede ir a hacer la compra por uno, y un hombre mecánico que camine por el cuarto probablemente no llegará ni siquiera hasta la tienda de la esquina sin llevarse por delante algún poste de alumbrado. Cada uno de estos hombres mecánicos era definitivamente una máquina: típicamente, llevaban a cabo una limitada

Colección Kobal

Estrella cinematográfica
El clásico de la ciencia-ficción *Metropolis* (1926), de Fritz Lang, ha influido en el público y en los productores de películas durante décadas, y ha plasmado en La Máquina, primera estrella robot de la historia del cine, las difusas imágenes del progreso

gama de acciones que no exigían la toma de ninguna decisión, y no parecían poseer ningún tipo de inteligencia.

Los robots en la literatura

Pero si los inventores y los ingenieros estaban varados por falta de ideas, los escritores de ciencia-ficción ciertamente no experimentaban estas restricciones creativas. La ciencia-ficción se ha enriquecido con la idea de los robots. De hecho, la propia palabra *robot* procede de una obra literaria. En 1923 el dramaturgo checoslovaco Karel Čapek (1890-1938) escribió una pieza teatral titulada *R.U.R.*; el título correspondía a las siglas de *Rossum's universal robots* (Los robots universales de Rossum). La obra trataba de la invención de un hombre mecánico tan perfecto que era capaz de llevar a cabo trabajos que tradicionalmente sólo podía realizar un ser humano. Finalmente los robots descubrían que no necesitaban a los hombres para nada, lo que dejaba a éstos en una situación poco airosa. En checo la palabra *robota* significa simplemente "trabajo". Por consiguiente, el título de la obra de Čapek se debería haber traducido como "Los trabajadores universales de Rossum"; pero, como quiera que fuera, la palabra "robot" prendió y desde entonces se ha convertido en el vocablo que se aplica a cualquier hombre mecánico que posea habilidades humanas.

Las fantasías de ciencia-ficción acerca de creaturas construidas para asemejarse a los seres humanos se remonta a la famosa novela gótica de Mary Shelley, *Frankenstein* (1818). Si bien no era mecánico, el monstruo creado por Victor Frankenstein estaba conformado a partir de un conjunto de elementos, aun cuando éstos se hubieran obtenido mediante el proceso más bien horripilante de profanar tumbas. Los seres invasores de *La guerra de los mundos* (1898), de H. G. Wells, eran, al menos en parte, mecánicos.

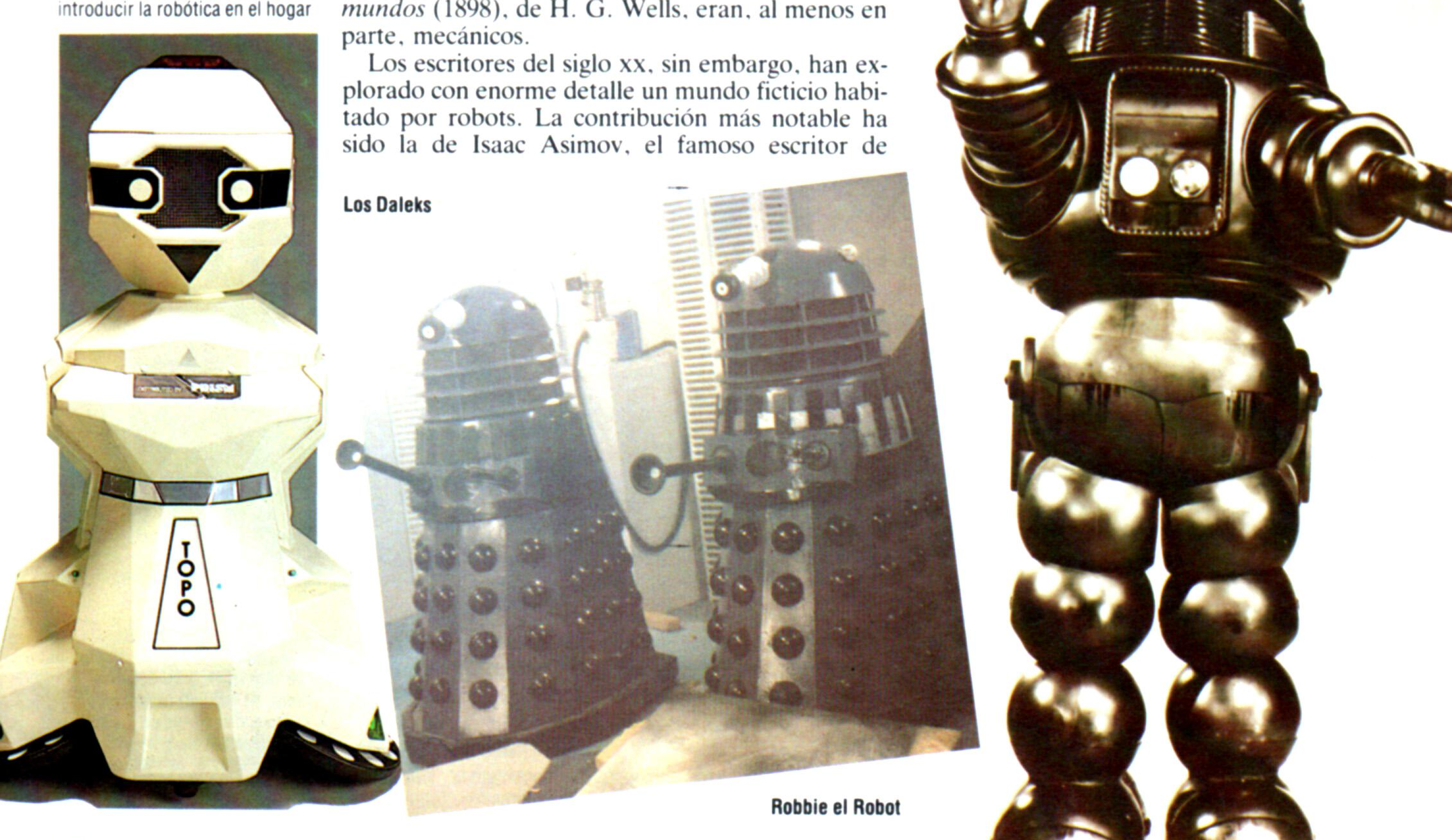
Los escritores del siglo XX, sin embargo, han explorado con enorme detalle un mundo ficticio habitado por robots. La contribución más notable ha sido la de Isaac Asimov, el famoso escritor de ciencia-ficción que comenzó su carrera en 1940 escribiendo relatos sobre robots y sus imaginables problemas operativos. El mundo robótico visionario de Asimov es tan completo que incluso ha formulado las tres "leyes de la robótica". De acuerdo a Asimov, las leyes están contenidas en el *Manual de robótica* (56.ª edición, 2058 d.C.). Evidentemente, el escritor ha concedido un margen de tiempo muy razonable antes de que los robots se conviertan en algo cotidiano.

En el cine y la televisión, los robots también han dejado su huella novelesca. La serie de televisión *Dr. Who* está densamente poblada por Daleks y Cybermen, y en las películas de la trilogía de *La guerra de las galaxias* C3P0 y R2D2 están en el mismo nivel de importancia que ios protagonistas humanos.

En comparación con estos vuelos de la fantasía, la utilización actual de los robots parece bastante trivial. Los robots industriales de las cadenas de montaje de automóviles reciben en la actualidad la mayor parte de la atención. Se calcula que este año 1985 habrá 25 000 en uso en la industria japonesa, 15 000 en Estados Unidos y 8 000 en la República Federal de Alemania. Se espera que la expansión del mercado europeo para robots industriales siga creciendo sin pausa.

No obstante, muchas personas opinan que los robots industriales son bastante torpes. Una máquina que suelda repetidamente componentes en la carro-

Realidad y fantasía
Los robots más famosos de la televisión europea deben de ser los Daleks. En realidad éstos son armaduras conducidas por personas, controladas desde su propio interior por sus creadores.
Robbie el Robot, de la película *Planeta prohibido*, constituye un buen representante del robot atento y sensible de la leyenda antropomórfica.
Topo *(abajo)*, robot doméstico ya retirado del mercado, fue un intento más o menos serio por introducir la robótica en el hogar



Los Daleks

Robbie el Robot

cería de un coche, o que atomiza pintura sin parar, apenas si representa la imagen novelística del robot. Que el robot ideal llegue a construirse alguna vez ya es una cuestión de conjeturas. Y si tal robot se diseñará a la imagen y semejanza del hombre es igualmente difícil de determinar. Pero examinando más de cerca algunos de los aspectos de la robótica, como haremos en esta serie de capítulos, podremos juzgar por nosotros mismos qué forma podrán tener los robots del futuro.

Glosario de robótica

Los robots, al menos en la ficción, han recibido tal número de denominaciones que tal vez le resulte útil contar con un glosario de los términos más comunes utilizados. Tenga presente, no obstante, que el mero hecho de que a algo se le dé un nombre ¡no implica necesariamente que exista!

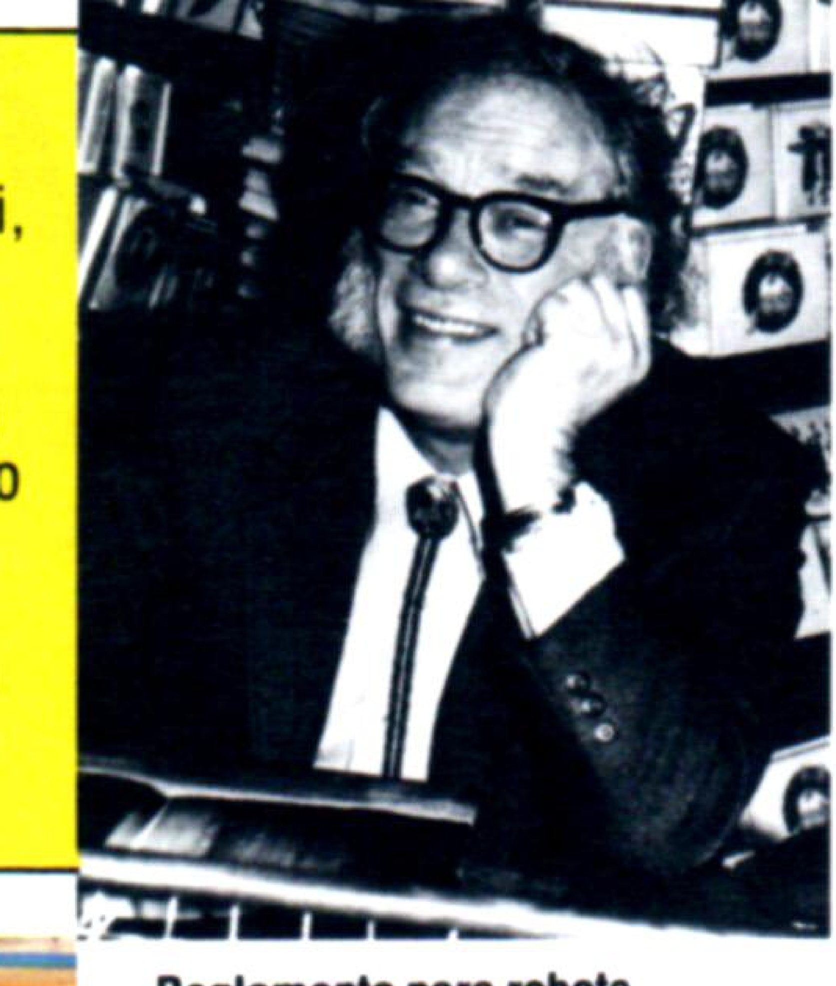
Androide: Robot diseñado para parecerse en todos los aspectos a un ser humano.
Antropomórfico: Literalmente, "de forma humana". Un androide es antropomórfico en todos los sentidos, pero muchos robots están diseñados para ser antropomórficos sólo en algunos aspectos. Por ejemplo, pueden tener un brazo que sea parecido a un brazo humano.
Automatización: Control automático de un proceso de fabricación.
Automatón: Máquina con mecanismos ocultos que normalmente realiza sólo una serie preestablecida de funciones. Los primeros hombres mecánicos eran autómatas. Asimismo, posee un significado más técnico cuando se asocia con *teoría de autómatas*, sistema analítico en virtud del cual se puede estudiar y describir cualquier dispositivo: robots, ordenadores, incluso personas.
Cibernética: Estudio de los sistemas de control y comunicaciones. Ideada por Norbert Weiner en 1947, el reclamo central de la cibernática es que se la puede utilizar para examinar sistemas biológicos como si fueran máquinas.
Cibert: Idea de ficción de un humanoide mecánico.
Ciborg: *Org*anismo *cib*ernético en el cual algunas partes son biológicas y otras mecánicas.
Cibot: Otra idea de ficción; robot con habilidades mentales humanas.
Doppelgänger: Réplica exacta de una determinada persona viviente, aunque por lo general sólo se trata de un espíritu o fantasma.
Droide: Robot bueno que obedece las tres leyes de Asimov.
Efector final: Terminología corriente para la "mano" de un robot.
Homunculi: Pequeños hombres o enanos.
Manipulador: Mano de un robot.
Mecanización: Sustitución de un proceso por un proceso mecánico.
Robot: Máquina capaz de llevar a cabo algunas funciones humanas.
Robótica: Ciencia que estudia los robots.
Trabajadores de cuello metálico: Robots industriales. A los oficinistas humanos a menudo se los llama trabajadores de "cuello blanco", y a los trabajadores manuales se los conoce como trabajadores de "cuello azul". Inevitablemente, se ha comenzado a aludir a los robots como trabajadores de "cuello metálico".

Las tres leyes de Asimov

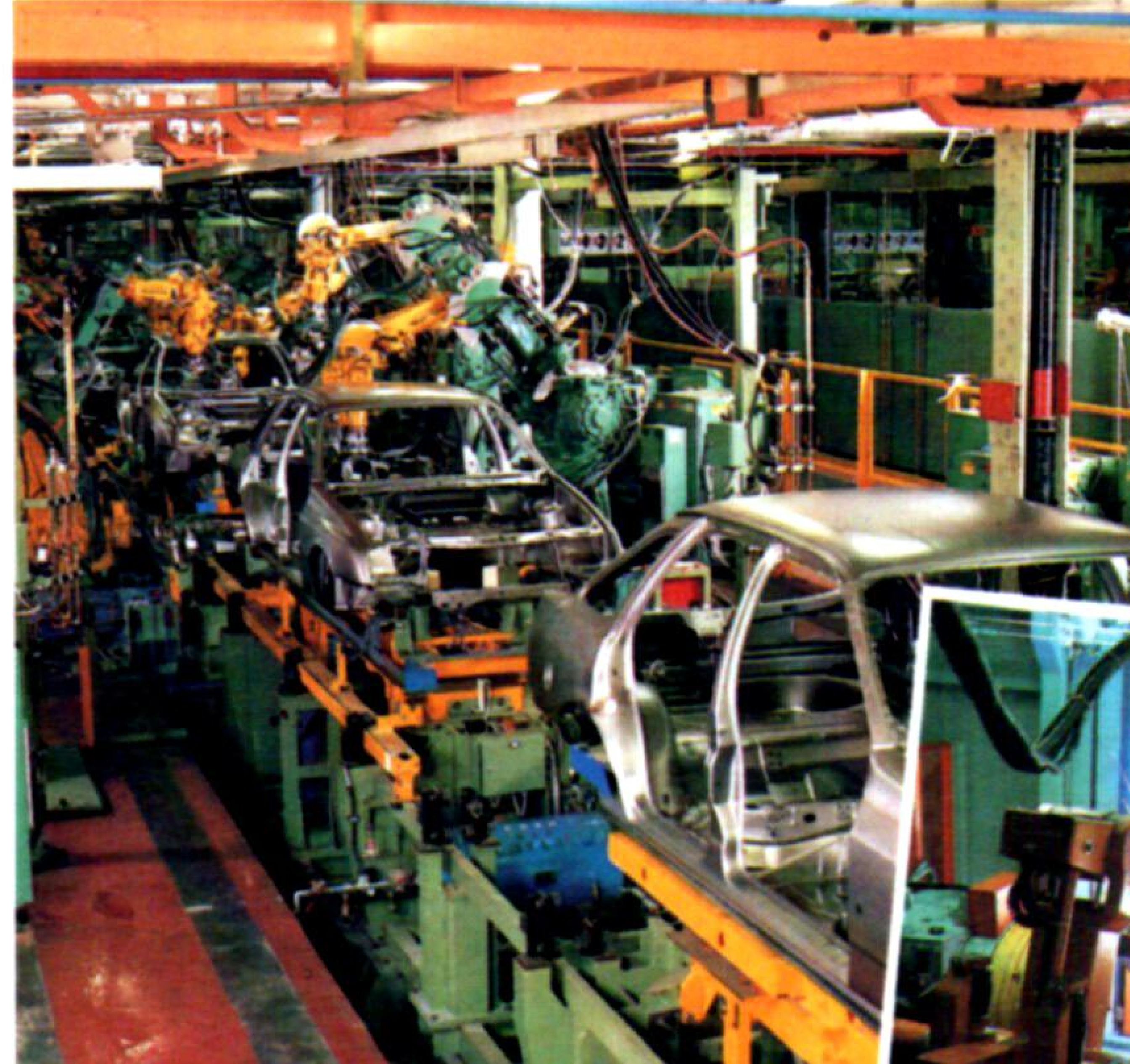
1. Un robot no ha de herir a ningún ser humano ni, por omisión, ha de permitir que se haga daño.

2. Un robot debe obedecer las órdenes que le son impartidas por los seres humanos, excepto cuando tales órdenes infrinjan la primera ley.

3. Un robot debe proteger su propia existencia en la medida en que dicha protección no infrinja la primera y la segunda ley.



Reglamento para robots
Cuando los robots sean capaces de emprender acciones independientes, las leyes de Asimov podrían constituirse en el fundamento de su conducta. Pero aún son incapaces de identificar a un ser humano, y la regulación jurídica de la interacción robot-hombre todavía es inaplicable



Ford: cadena de montaje del Sierra

Fiat: montaje del eje de torsión

Montaje robot
Durante algún tiempo más, los robots se utilizarán básicamente en las cadenas de producción. La economía de la producción masiva hace de ellos unos trabajadores ideales para cadenas de montaje, tal como muestran claramente las factorías de Fiat y Ford. La especialización suele exigir que estos robots se reduzcan a uno o dos brazos equipados con pinzas, llave de tuercas y aparatos para soldar

¿Qué es un robot?

Si busca en el glosario, verá que hemos definido a un robot como "una máquina capaz de llevar a cabo algunas funciones humanas", si bien su aspecto no necesariamente parece humano. Obviamente, ésta es una definición muy amplia; se podría aplicar, por ejemplo, a los ordenadores (porque realizan algunas funciones de cálculo). No obstante, en el uso común, un robot tendría algunas cualidades humanas reconocibles. Sería capaz de desplazarse, o quizá incluso de caminar. Podría tener un brazo que se asemejara a un brazo humano. Estaría en condiciones de ver y oír cosas. Hasta podría poseer un alto grado de inteligencia.

La forma y las habilidades exactas de los robots dependen fundamentalmente de dos factores: lo que deseamos que hagan y lo que podemos conseguir que hagan. Por ejemplo, un robot industrial utilizado para soldar puede no ser capaz de desplazarse, no porque nosotros no podamos hacer que se desplace, sino porque deseamos que permanezca en un sitio y se dedique a cumplir con la misión asignada. Del mismo modo, un robot doméstico podría ser capaz de preparar una taza de té, pero quizá no pudiera subir las escaleras para llevársela a usted hasta su dormitorio.

El término "robot" se ha convertido en la palabra genérica para todas las máquinas parecidas a los humanos, y los límites acerca de lo que son y de lo que pueden hacer conciernen a quienes los diseñan y los construyen.

Actuación repetida

El LOGO utiliza la técnica matemática de la recursión para crear complejos diseños mediante sencillas instrucciones



Uno de los primeros programas que definimos en el curso era un procedimiento para dibujar un cuadrado. La definición le indicaba a la tortuga que avanzara una cierta distancia, que girara 90º a la derecha y que repitiera esos dos pasos tres veces más. Ésta es otra forma de dibujar un cuadrado:

```
TO CUADRADO
  FD 50
  RT 90
  CUADRADO
END
```

Al ejecutar esto, la tortuga dibujará un cuadrado y luego seguirá desplazándose a lo largo del perímetro del cuadrado hasta que se interrumpa con Control-G o BREAK. Lo más destacable de este nuevo procedimiento CUADRADO es que se llama a sí mismo; en otras palabras, que es *recursivo*.

Cuando se ejecuta este procedimiento, el LOGO va a buscar la definición de CUADRADO y empieza a obedecer las instrucciones. La tortuga se mueve FORWARD 50 y luego gira RIGHT 90. La siguiente instrucción es CUADRADO, de modo que el LOGO carga a memoria la definición de CUADRADO y empieza a ejecutarla. Esto se producirá *ad infinitum* si no se interrumpe el programa.

También se pueden utilizar llamadas recursivas en procedimientos que exijan inputs:

```
TO POLI :LADO :ANGULO
  FD :LADO
  RT :ANGULO
  POLI :LADO :ANGULO
END
```

Este procedimiento puede producir todos los polígonos que hemos definido hasta ahora en el curso (véase p. 1025), así como muchos de los que no nos hemos ocupado (pruebe a utilizar este procedimiento con un valor de ángulo de 89). También es posible cambiar el valor de la entrada en la llamada recursiva. De esta manera:

```
TO POLIESPI :LADO :ANGULO
  FD :LADO
  RT :ANGULO
  POLIESPI (:LADO+5) :ANGULO
END
```

La única diferencia entre este procedimiento y POLI es que se le suma 5 al valor de LADO cada vez que se lo llama. De modo que si se empieza con POLIESPI 10 90, la primera llamada dibujará una línea de longitud 10, la segunda será de 15, luego 20, etc. El resultado será una espiral. Quizá le agrade experimentar con entradas diferentes: 10 90, 10 95, 10 120, 10 117, 10 144 y 10 142 son buenos puntos de partida. También puede tratar de modificar el procedimiento: una posibilidad es cambiar la suma por resta o multiplicación.

El siguiente es un procedimiento similar que incrementa el ángulo en vez del valor del lado:

```
TO ENESPI :LADO :ANGULO :INC
  FD :LADO
  RT :ANGULO
  ENESPI :LADO (:ANGULO + :INC):INC
END
```

Pruebe diversas inputs: 5 0 7, 10 40 30, 15 2 20, 5 30 20 servirán para empezar. ¿Por qué algunas formas se cierran y otras no? ¿Puede descubrir alguna regla?

La simple repetición de un trozo de código se denomina *iteración*. Con esta finalidad el LOGO utiliza REPEAT (repetir), mientras que otros lenguajes emplean una variedad de construcciones, como FOR...NEXT, REPEAT...UNTIL y WHILE...WEND. Sin embargo, el LOGO se basa mucho más en la recursión que en la iteración. Si ha programado en otros lenguajes puede que tenga cierta dificultad para evitar la utilización de la iteración, pero los gráficos tortuga son ideales para experimentar con llamadas recursivas.

Reglas para interrupción

Todos los procedimientos recursivos que hemos analizado hasta ahora continúan repitiéndose indefinidamente. Es evidente que necesitamos una manera de hacer que un procedimiento se detenga en algún punto. Tomando como ejemplo el procedimiento CUADRADO, un posible lugar para interrumpirlo sería después de que hubiera dibujado un cuadrado completo y el encabezamiento de la tortuga estuviera otra vez en 0. Esto se puede hacer agregándole al procedimiento un "método de parada":

```
TO CUADRADO :LADO
  FD :LADO
  RT 90
  IF ENCABEZAMIENTO=0 THEN STOP
  CUADRADO :LADO
END
```

Las nuevas primitivas son STOP e IF. La primera de estas instrucciones hace que la ejecución de un procedimiento se interrumpa y el control retorne al procedimiento que lo llamó. Una sentencia IF es la forma del LOGO de tomar decisiones. IF va seguido de una condición, y THEN por una acción que se efectúa si la condición se cumple.



Veamos una versión de POLIESPI con un método de parada y consideremos exactamente qué sucede cuando se ejecuta:

```
TO POLIESPI :LONGITUD
  IF :LONGITUD > 15 THEN STOP
  FD :LONGITUD
  RT 90
  POLIESPI (:LONGITUD + 5)
END
```

Veamos qué es lo que sucede cuando ejecutamos POLIESPI 10. Se llama al procedimiento POLIESPI y se define una variable local inicializándola a 10. Puesto que este valor no es mayor que 15, el LOGO procede a ejecutar el movimiento FD 10 RT 90 y después efectúa una nueva llamada a POLIESPI, pero esta vez con un valor de entrada de 15. Esto hace que se vuelva a llamar a una copia del procedimiento. Dado que LONGITUD no es mayor que 15, la tortuga se mueve FD 15 RT 90 y se realiza otra llamada a POLIESPI. Pero esta vez la variable local se ha incrementado a 20, de modo que el procedimiento se interrumpe y el control retorna al procedimiento que lo llamó (POLIESPI 15). Este procedimiento, a su vez, ha llegado a su línea final y le devuelve el control a su procedimiento de llamada. Éste también se interrumpe, en cuyo momento el programa ha llegado a su final.

Hemos visto cómo en LOGO la recursión implica procedimientos que llaman a copias de sí mismos. Es importante tener presente que las llamadas recursivas son copias que existen del procedimiento original, trabajando como si fueran completamente independientes del mismo. Al acabar, dicho procedimiento siempre le devuelve el control al procedimiento que lo llamó. Para ilustrar más claramente el proceso de retornar desde llamadas a procedimientos, podemos reescribir POLIESPI de la siguiente manera:

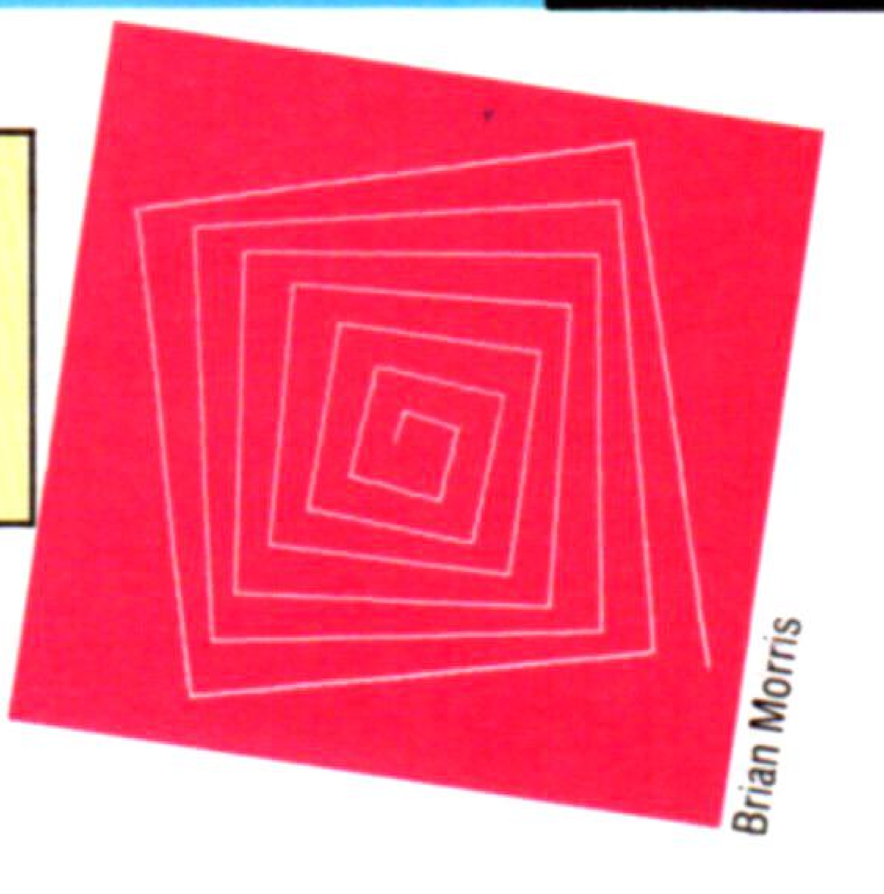
```
TO POLIESPI :LONGITUD
  IF :LONGITUD > 15 THEN STOP
  POLIESPI (:LONGITUD + 15)
  FD :LONGITUD
  RT 90
END
```

Al ejecutarlo verá que el programa hace su dibujo "hacia atrás": las líneas se dibujan en espiral hacia adentro en vez de hacia afuera. (Esto se apreciará más claramente si se utiliza en la sentencia de condición un valor mayor: por ejemplo, utilizando 50 en lugar de 15.) Lo que es significativo aquí es que el LOGO dibuja cada línea cuando se devuelve el control desde las llamadas a procedimientos. En nuestros ejemplos anteriores se dibujaba una línea y el control pasaba entonces a otro procedimiento. Pero aquí todos los procedimientos son llamados antes de empezar ningún dibujo, y el último valor creado de LONGITUD es el que se utiliza primero.

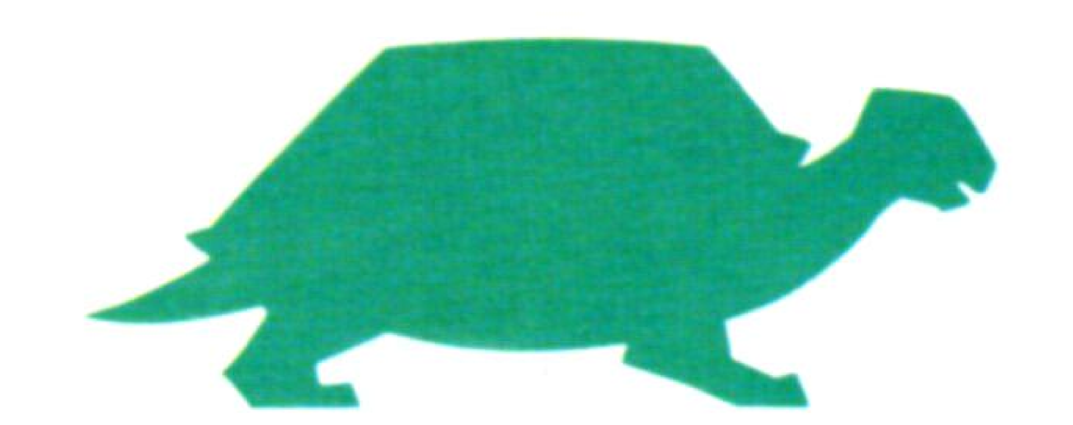
Por último, debemos observar que la recursión es una técnica que ocupa muchísima memoria. Los procedimientos en los cuales la llamada recursiva está en la última línea son los que se implementan más eficazmente, dado que no ocupan ninguna memoria adicional independientemente de cuántas veces sean llamados. Si se puede escribir un procedimiento de modo que sea "recursivo al final", por lo general siempre vale la pena hacerlo.

Procedimientos (3)

Escribir un procedimiento recursivo para dibujar una torre de cuadrados uno encima del otro, reduciendo cada vez la longitud del lado a la mitad



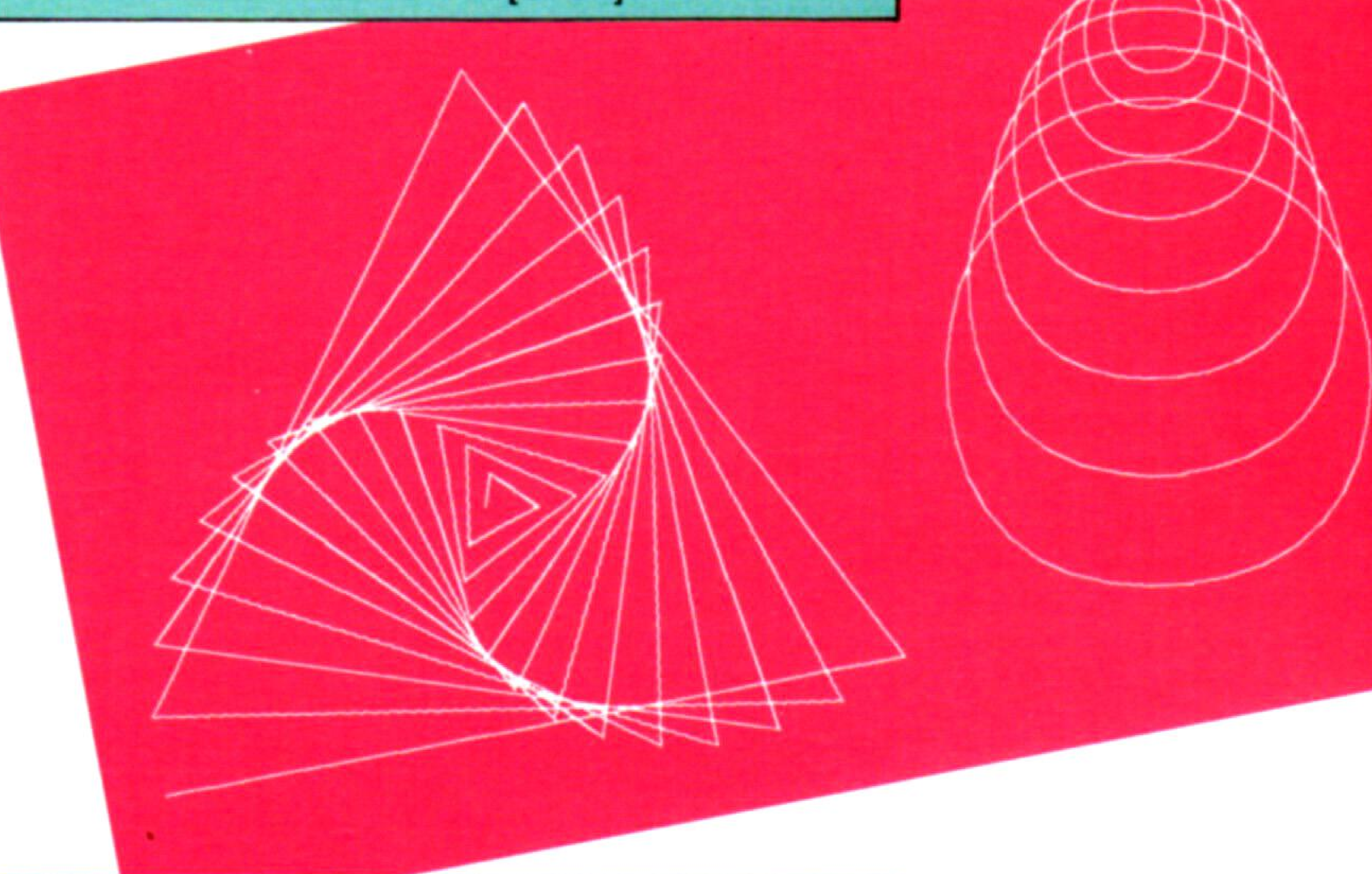
Brian Morris



Complementos al LOGO

En todas las versiones LCSI, la parte de acción de una sentencia IF se escribe como una lista dentro de corchetes y sin el THEN. Por ejemplo:

```
IF ENCABEZAMIENTO = 0[STOP]
```



Respuestas a ejercicios

Un procedimiento para dibujar un círculo, dando el radio como entrada, con la posición actual como un punto de la circunferencia:

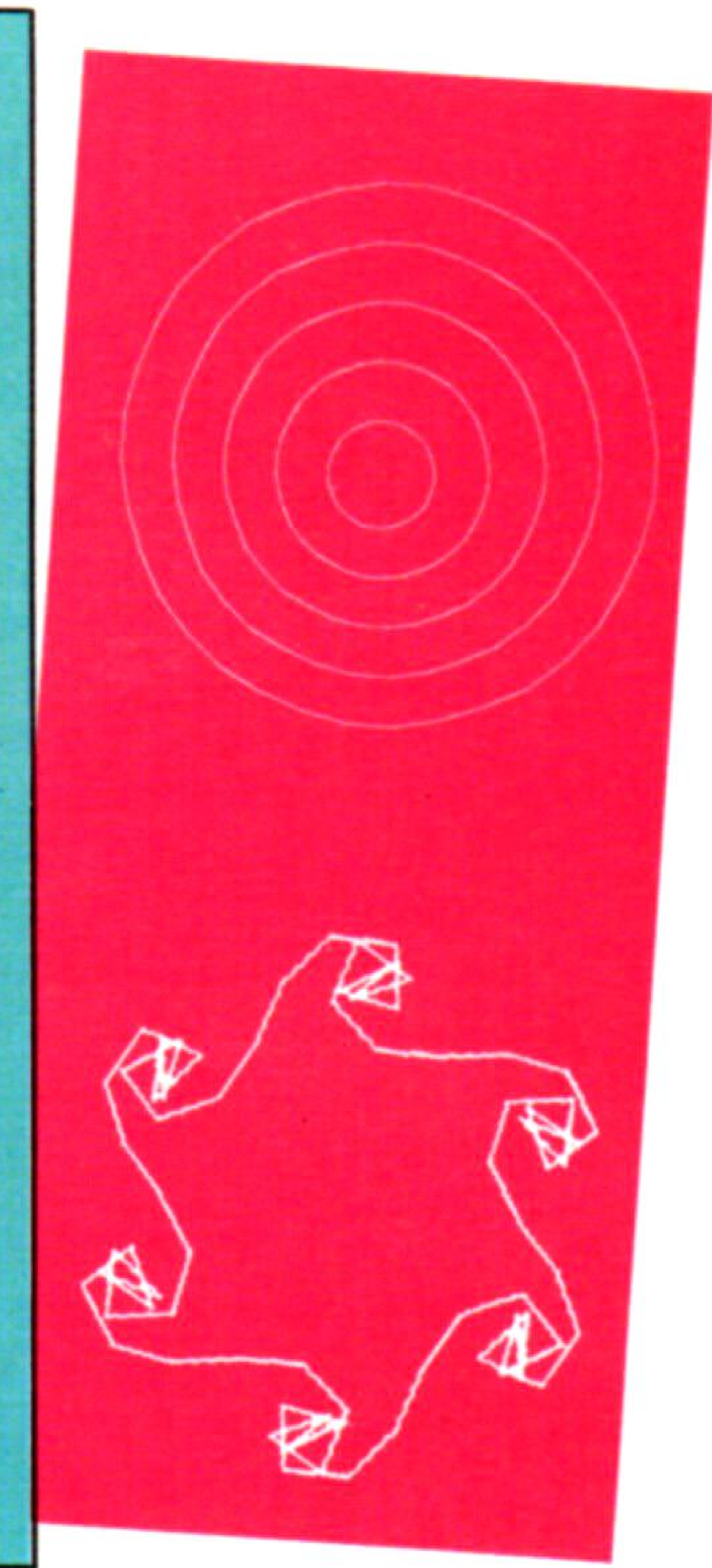
```
TO CIRCUITO :RADIO
  REPEAT 36 [FD (2*:PI*:RADIO/36) RT 10]
END
MAKE "PI(3.14159)
```

Si adaptamos el procedimiento para que el centro del círculo esté en la posición actual, tendremos:

```
TO C.CIRCULO :RADIO
  PU LT 90 FD :RADIO RT 90 PD
  CIRCULO :RADIO
  PU LT 90 BK :RADIO RT 90 PD
END
```

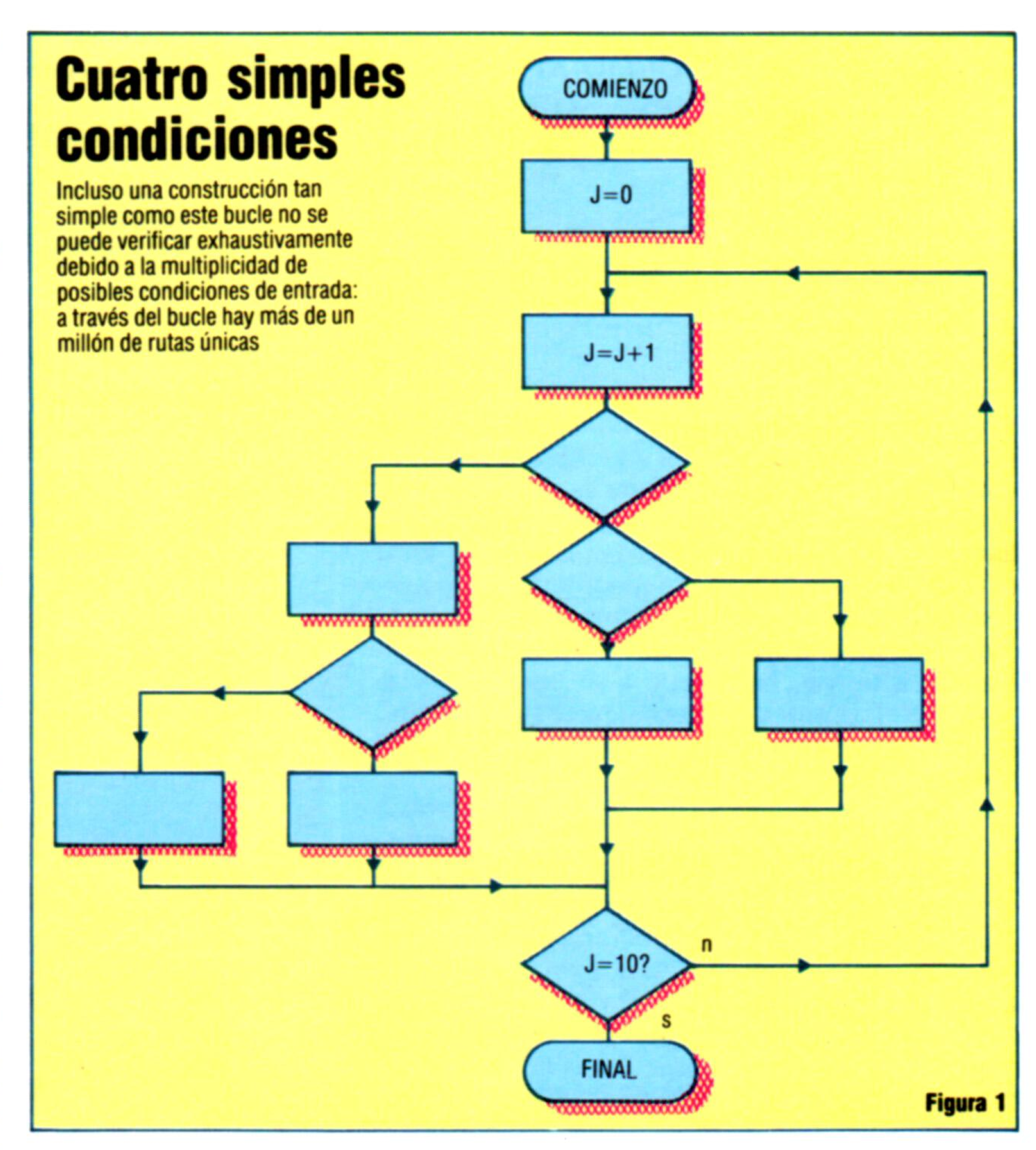
BLANCO utiliza C.CIRCULO para dibujar 5 círculos concéntricos:

```
TO BLANCO
  C.CIRCULO 10 C.CIRCULO 20 C.CIRCULO 30
  C.CIRCULO 40 C.CIRCULO 50
END
```



Comprobación de rutinas

En este capítulo final analizamos los métodos que se pueden utilizar para probar un programa ya terminado



Cuatro simples condiciones

Incluso una construcción tan simple como este bucle no se puede verificar exhaustivamente debido a la multiplicidad de posibles condiciones de entrada: a través del bucle hay más de un millón de rutas únicas

Una de las grandes ventajas de programar en un lenguaje interpretado como el BASIC es que el código se puede verificar a medida que se va escribiendo. El programador puede en cualquier momento entrar RUN y ver lo que sucede. En la mayoría de las máquinas, es muy fácil interrumpir la ejecución de un programa, imprimir (PRINT) los valores de variables clave, modificar estos valores y después CONTinuar. Todo esto significa que se pueden detectar y corregir la mayoría de las equivocaciones. Pero esta especie de depuración *ad hoc* no es un sustituto de la comprobación formal, que se debe realizar cuando el programa se haya completado.

La prueba definitiva pretende asegurar que un programa hará exactamente lo que está destinado a hacer. Para cualquier conjunto posible de datos de entrada correctos debe producir la salida correcta, y para toda entrada ilegal debe emprender las acciones apropiadas. Una forma sencilla de verificar un programa podría ser la de darle una muestra de todas las entradas legales posibles y comprobar después que los resultados son los esperados. Sin embargo, para la mayoría de los programas esto será inviable. Imaginemos un programa que tome dos enteros, los sume e imprima el resultado. Siguiendo este método, habrá de ser ejecutado ¡para todos los valores enteros posibles! Y ésta es sólo una parte del problema, puesto que también habrá de probarse con todos los valores *ilegales*.

Otra posibilidad podría ser examinar todos los "caminos" a través del programa. Se puede descubrir un camino determinado siguiendo una ruta a través de un diagrama de flujo desde el principio hasta el final. Cada bifurcación que aparezca en el recorrido abre caminos alternativos y cada bucle va agregando más. La figura 1 muestra un programa que es un bucle que contiene unas cuantas sentencias IF...THEN. Dentro del cuerpo del bucle hay cuatro caminos y el bucle se ejecuta 10 veces. Esto significa que la cantidad de rutas exclusivas desde "comienzo" a "final" es de 1 398 100: una cifra asombrosa para lo que probablemente no consista en nada más que en una docena de líneas de código.

Por consiguiente, si la comprobación exhaustiva de datos no funciona y la comprobación a fondo de la lógica tampoco, ¿qué es lo que sí funciona? La respuesta, sorprendente, es nada. No existe ninguna forma de probar completamente un programa complejo en un tiempo razonable. En parte por esta razón, la comprobación sigue la ley de que la cantidad de errores hallados por unidad de esfuerzo disminuye con cada unidad extra. En consecuencia, el momento adecuado de dejar de buscar errores es cuando el esfuerzo de hacerlo importa más que el costo de los fallos aún no detectados.

Sin embargo, a pesar de estos inconvenientes, vale la pena desarrollar algún método de comprobación. Una presunción razonable es que si una máquina opera correctamente con un dato de un tipo determinado, operará correctamente con todos los datos del mismo tipo. De modo, pues, que si una subrutina funciona para un entero positivo dentro de su escala, deberá funcionar para todos los enteros positivos de esa escala. Esto nos lleva a una clase de comprobación conocida como *comprobación de clases de equivalencia*. La idea consiste en desarrollar un conjunto de casos de prueba cada uno de los cuales sea representativo de una clase de casos que se comportarán de la misma manera. Por lo tanto, si un trozo de código comprueba que una entrada está comprendida entre 1 y 100, deberemos verificar las entradas que sean menores que el menor valor esperado, mayores que el mayor valor esperado, y que se encuentren dentro de la escala esperada (valor < 1, valor > 100, y 1= < valor= < 100).

El examen de cada uno de los caminos lógicos también se puede simplificar llamando a cada rutina por todos sus puntos de entrada (si bien idealmente debería haber sólo uno) y, dentro de cada rutina, cubriendo cada una de las posibles salidas de todas las decisiones. En la figura 2 tenemos una rutina para ajustar los puntos de premio de un juego. Toma los parámetros de entrada PREMIO,

Sólo probando
Un juego completo de datos de prueba calculados de antemano para el ejemplo ilustrado en los diagramas de flujo podría tener el siguiente aspecto:

	ENTRADA		SALIDA
NIVEL	ACIERTOS	PREMIO	PREMIO
6	10	200	1300
4	10	550	2300
7	10	550	3950
4	10	200	800
7	10	200	1400
1	20	2500	2600
1	20	550	550
6	5	200	300
6	50	200	300
4	5	2500	2600
7	50	2500	2600
4	50	550	550
7	5	550	550



NIVEL y ACIERTOS y devuelve un valor (posiblemente nuevo) para PREMIO. Se podría escribir así:

```
6030 IF NIVEL > 2 AND ACIERTOS=10 THEN
     PREMIO=PREMIO*NIVEL
6040 IF NIVEL=6 OR PREMIO > 2000 THEN
     PREMIO=PREMIO+100
```

Para cubrir la salida de cada expresión condicional necesitamos considerar las entradas a cada una que producirían una salida de "sí" o "no". En ambas decisiones estamos observando los efectos de dos variables combinadas mediante un operador lógico (AND y OR). Ello significa que tenemos que tomar en consideración los valores combinados de las variables y no sus valores individuales. Para ver por qué, consideremos lo que sucedería si comprobáramos valores para NIVEL de 4 y 1 y para ACIERTOS de 10, 5 y 20 en la primera decisión. Cuando NIVEL=4, se comprueban los tres valores de ACIERTOS, pero cuando NIVEL=1 no son comprobados. Esto es un caso en el cual una parte de una decisión "enmascara" a otra. Para comprobar cada parte por separado, es mejor simplificar las decisiones compuestas.

Observando la figura 3, podemos ver que con cuatro decisiones binarias hay 2^4 (=16) posibles resultados y debemos cubrirlos todos. Un punto de partida es hacer la lista de las condiciones para un resultado sí o no para cada decisión, como ésta:

	1	2	3	4
sí	NIVEL>2	ACIERTOS=10	NIVEL=6	PREMIO>20000
no	NIVEL=2 NIVEL<2	ACIERTOS<10 ACIERTOS>10	NIVEL<6 NIVEL>6	PREMIO=2000 PREMIO<2000

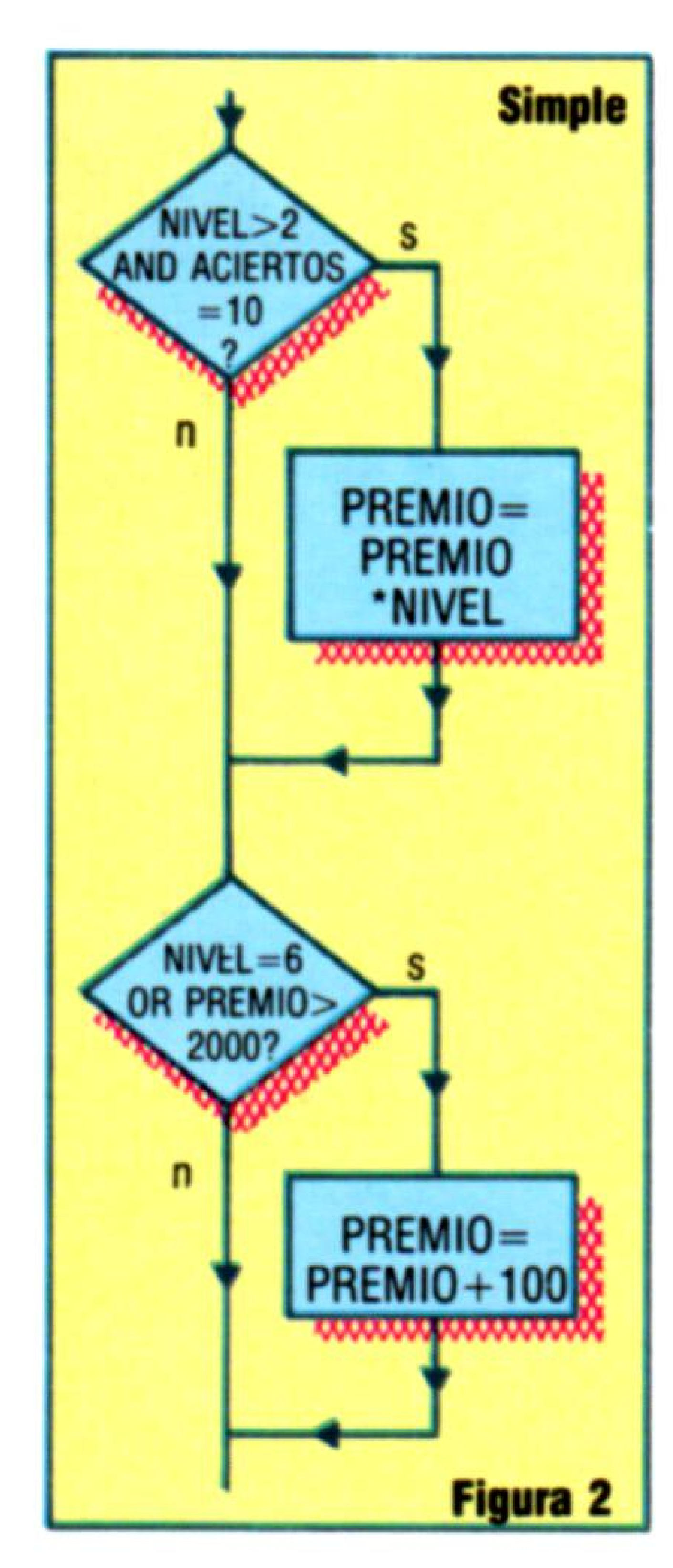
Las mismas se pueden utilizar para deducir los valores para datos de prueba representativos. Por ejemplo, para el camino adfi (véase fig. 3), NIVEL debe ser mayor que 2 e igual a 6, ACIERTOS no debe ser igual a 10 y PREMIO puede tener cualquier valor (porque no está implicado). Los valores NIVEL=6, ACIERTOS=20 y PREMIO=150 emplearían este camino (al igual que muchos otros, por supuesto). La ruta abehj se podría probar con NIVEL=4, ACIERTOS=10 y PREMIO=600 (no olvide que estamos hablando del valor de *entrada* de PREMIO que posteriormente puede ser multiplicado por NIVEL).

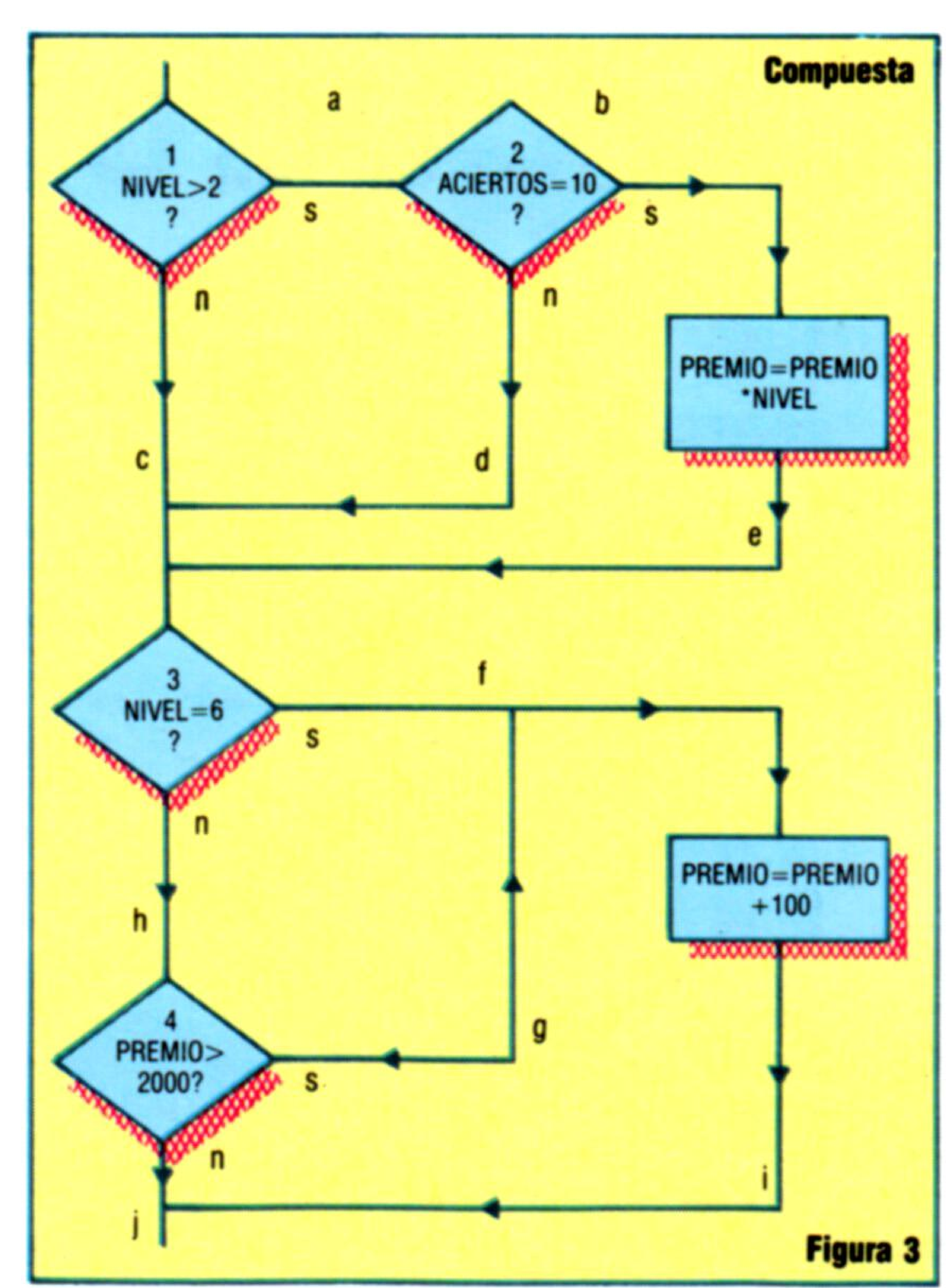
Es igualmente importante el hecho de que los resultados que debe producir cada uno de los conjuntos de datos se deben calcular antes de la ejecución de prueba, de modo que se puedan comparar los resultados. Los datos de entrada solos, únicamente probarán si el programa funciona. Para comprobar que esté efectuando lo que debe hacer, la salida se debe calcular a mano previamente. Podemos ver un conjunto completo de casos de prueba para este ejemplo (izquierda).

Ya en posesión de un método para "ejercitar" nuestro software, ahora necesitamos una forma de abordar un programa más largo de modo que la complejidad no resulte abrumadora. Es aquí donde se percibe otro de los beneficios de la programación estructurada. Los programas escritos como un conjunto de módulos independientes dispuestos según una jerarquía nos permiten probar cada módulo de forma individual. Dado que los módulos están dispuestos de esta manera, podemos empezar por el módulo superior e ir trabajando hacia abajo, comprobando cada módulo individual sólo cuando ya se han verificado todos los que hay por encima del mismo, y podemos utilizar módulos ya verificados para proporcionar datos a los módulos inferiores de la estructura.

El módulo que se esté probando tendrá por encima (a menos que sea el primero), un módulo *conductor* totalmente probado. Los módulos por debajo de él están todavía sin comprobar y por consiguiente no son fiables, de manera que se simulan mediante cortos trozos de código que simplemente devuelven los datos de prueba apropiados cuando son llamados por el módulo que se está verificando. Para usar esta técnica se suele emplear un esqueleto de código en el cual se pueden colocar las rutinas modulares para su comprobación. La figura 4 ilustra este principio. Los módulos 1, 2 y 3 ya se han comprobado, mientras que los módulos 5, 6 y 7 son simulados.

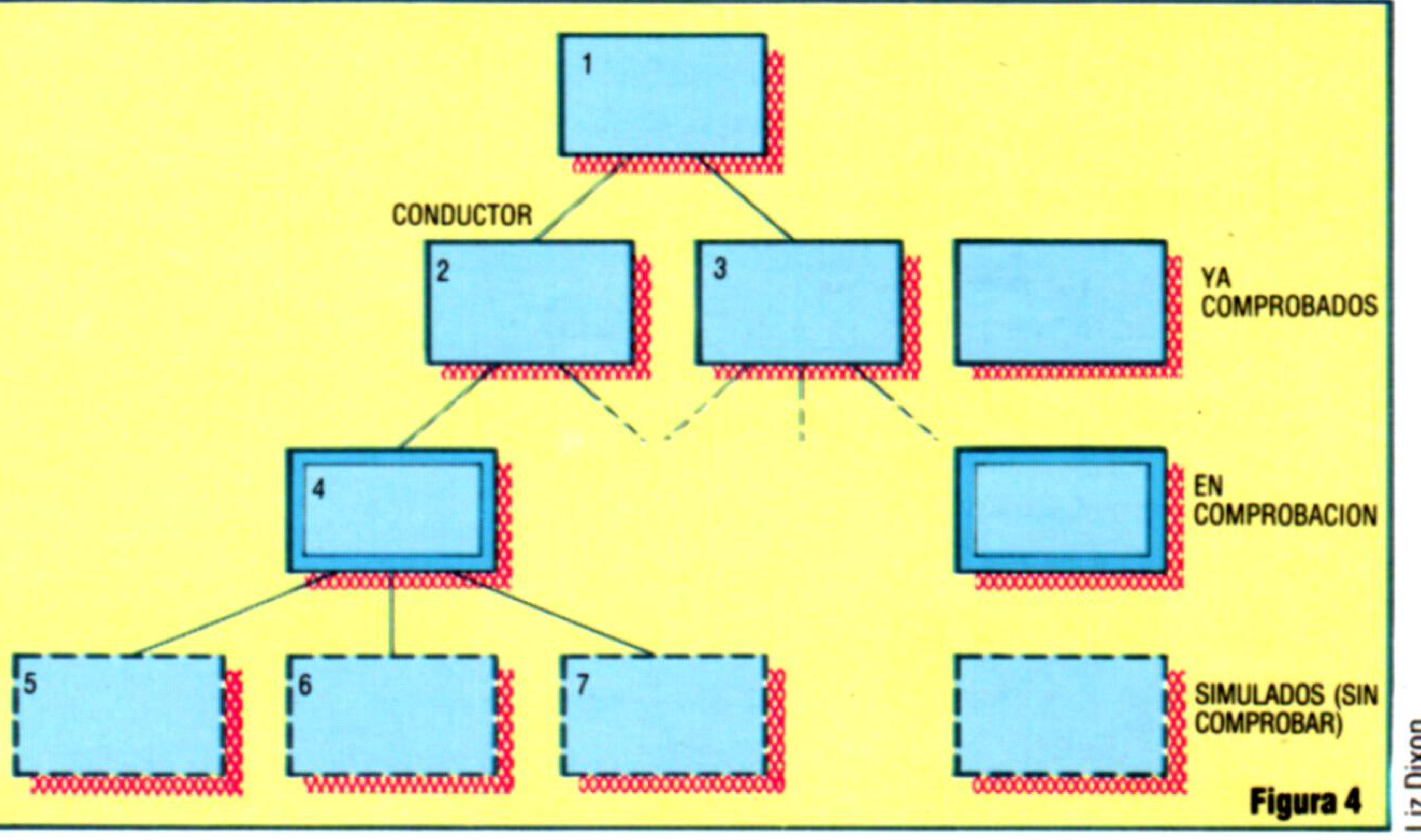
Debemos hacer hincapié en un último punto. La comprobación es una parte importante del ciclo vital del programa y, como tal, merece estar bien documentada.





Enmascarar decisiones
La simplificación de decisiones y el etiquetado de los enlaces ayudan a la verificación

Comprobación de arriba abajo (top-down)
La comprobación se simplifica muchísimo mediante el enfoque de arriba abajo, porque cada módulo se puede verificar después de ser escrito, tanto de forma aislada como en asociación con otros módulos ya comprobados. El comportamiento de los módulos aún no escritos se puede simular escribiendo pequeñas rutinas ficticias que generen artificialmente ejemplos de la salida prevista del módulo



Liz Dixon

Otro método de clasificación

INICIO
NÚM. DE ELEMENTOS
N
DIMENSIONAR TABLA CON N ELEMENTOS
T(N)
DESDE X=1 HASTA N
CARGAR LA TABLA
SIGUIENTE
M=N
PONER J CON VALOR 1
DESDE P=2 HASTA M
T(J) < T(P)?
NO
J=P
SÍ
SIGUIENTE
CAMBIAR ELEM. M POR ELEM. J
RESTAR 1 A M
NO
M IGUAL 1
SÍ
DESDE R=I HASTA N
VISUALIZAR ELEMENTO I DE TABLA
SIGUIENTE
END

He aquí otro eficaz procedimiento de clasificación, en que ésta se efectúa por el máximo

En el capítulo anterior examinamos la representación gráfica de dos métodos de clasificación válidos tanto para valores numéricos como alfanuméricos, los llamados métodos de "burbuja".

El empleo que presentamos en esta ocasión está realizado mediante la llamada técnica de clasificación por el máximo. Consiste en tomar una serie de datos, que, uno detrás de otro, se van disponiendo en el lugar correspondiente, siempre según su valor dentro de los elementos que componen la lista. Al comienzo de la misma quedan aquellos que tengan un valor más bajo; siguen los que tengan asignado un valor mayor, que deberá aumentar conforme vayan aproximándose al final.

Para poder realizar lo mencionado, deben efectuarse varias pasadas, comparándose los elementos. De esta manera, luego de una primera pasada el número de mayor valor quedará colocado en último lugar. En la segunda vuelta esta posición ya será ignorada, con lo cual se compararán los elementos desde el primero hasta el penúltimo, correspondiéndole ocupar este lugar al mayor de ellos. El proceso continúa hasta que todos los datos queden correctamente ordenados.

Se facilita un programa BASIC pensado para su uso en una máquina Commodore, basado en el diagrama sobre una lista de números introducidos al azar y que deben mostrarse ordenados de mayor a menor.

```
10 REM CLASIFICACION LISTA NUMEROS
20 INPUT "ENTRA NUM. ELEMENTOS TABLA:";N
30 DIM T(N)
40 FOR X=1 TO N
50 INPUT "NUMERO"; T(X)
60 NEXT X
70 M=N
80 J=1
90 FOR P=2 TO M
100 IF T(J)<T(P) THEN GOTO 120
110 J=P
120 NEXT P
130 C=T(M) : T(M)=T(J) : T(J)=C
140 M=M-1
150 IF M>1 THEN GOTO 80
160 FOR R=1 TO N
170 PRINT T(R)
180 NEXT
190 END
```

Portátil notable



Examinamos hoy el PX-8, una máquina "de regazo" con 64 Kbytes de RAM, CP/M y un conjunto de software

El PX-8 viene en una carcasa del tamaño de una guía telefónica y pesa aproximadamente 2,3 kg. El acabado de la carcasa es en dos tonos de beige, con un asa metálica escamoteable; a primera vista el paquete se parece muy poco a un ordenador. Sin embargo, parte de la carcasa se desliza para dejar al descubierto un teclado completo de ordenador y una pantalla de visualización hasta entonces oculta debajo. La pantalla se libera desplazando un mando etiquetado como "UNLOCK", que también deja al descubierto una grabadora de cintas de microcassette. El panel de la pantalla se puede colocar en cualquiera de 11 posiciones, aunque sólo cinco o seis proporcionan un buen ángulo visual.

El teclado tiene 72 teclas tipo máquina de escribir, codificadas por color para indicar su uso. Las teclas alfanuméricas oscuras están dispuestas de acuerdo al trazado QWERTY estándar en las versiones para Estados Unidos y Gran Bretaña (existe, asimismo, un modelo AZERTY francés a la venta en el resto de Europa), con el signo "£" en el teclado británico reemplazando el "#" (carácter numérico) de la versión norteamericana. (En realidad, desde cualquiera de los teclados se puede acceder a todos los caracteres "internacionales" cambiando los interruptores del PX-8. Este proceso se explica claramente en el manual del usuario.) También hay cuatro teclas de color naranja para control del cursor, teclas Insert, Delete y Home, tres teclas de funciones del sistema (Escape, Pause y Help) y cinco teclas de función programable.

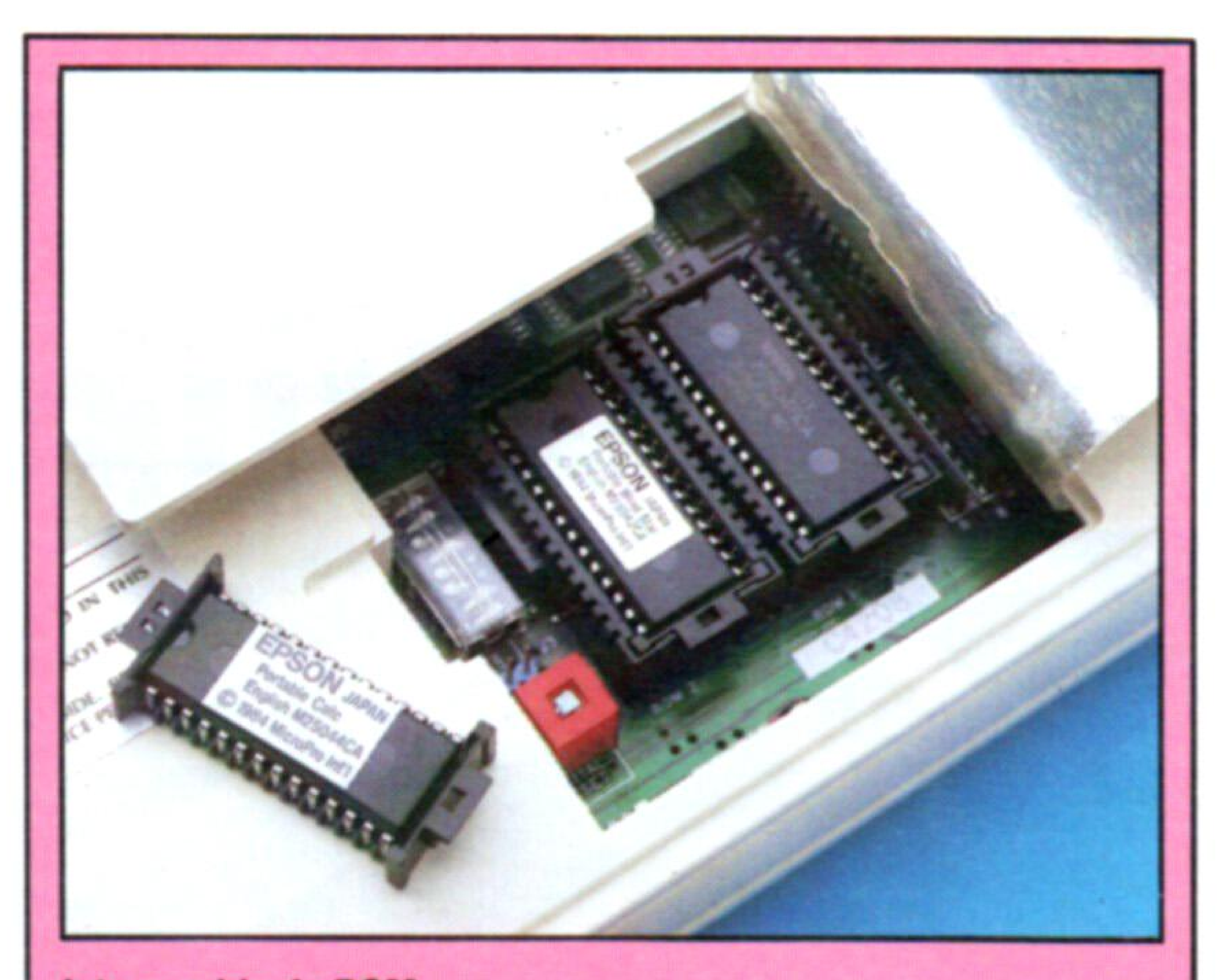
La fabricación del teclado responde a un estándar elevado y es especialmente fácil de utilizar sosteniendo la máquina en el regazo. Sin embargo, el teclado es menos útil cuando el PX-8 está colocado sobre el escritorio, dado que las teclas exigen una presión directa de arriba abajo. Se proporcionan dos patas retráctiles; éstas inclinan la unidad pero no llegan a solventar el problema. Las teclas Caps Lock, Number e Insert son interruptores de posición que se utilizan para cambiar las modalidades de visualización del PX-8: tres pequeños LED rojos indican cuál es la modalidad que está en operación en cada momento.

La documentación Epson es exhaustiva y está muy bien escrita. Se suministran dos gruesos manuales. El primero de ellos es un manual para el usuario de varios cientos de páginas, que abarca la instalación de la máquina, la utilización del hardware y el software y las operaciones de CP/M. Este manual también incluye mapas de memoria, listas completas de los caracteres disponibles y sus códigos, y un programa más bien largo en lenguaje má-

La última oferta
El ordenador "de regazo" PX-8 lo fabrica Epson, empresa famosa por sus impresoras matriciales, el portátil HX-20 y el ordenador de gestión para sobremesa QX10. El PX-8 viene con 64 K de RAM, una pantalla de visualización en cristal líquido (LCD), CP/M y varios paquetes de software



Una gran pantalla
El PX-8 tiene una pantalla LCD de 8 líneas de 80 caracteres que funciona en base a la batería de la máquina. La pantalla proporciona una resolución para visualizaciones gráficas de 480 por 64 pixels



Intercambio de ROM
Deslizando un pequeño panel situado en la cara inferior del PX-8 quedan al descubierto las ranuras que albergan el software basado en ROM. El Portable Wordstar está instalado en la máquina, junto con el sistema operativo CP/M

quina para guardar y cargar la pantalla de gráficos desde disco. El segundo volumen es una excelente guía de referencia de programación en BASIC, que también tiene varios centenares de páginas. Este libro empieza explicando cómo instalar y utilizar el BASIC (suministrado, como el software incluido, en una "cápsula" de ROM), antes de pasar a un claro análisis de la naturaleza de la programación, un examen de las diversas modalidades de visualización del PX-8 y un detallado análisis de todas las instrucciones de BASIC disponibles.

El PX-8 utiliza una CPU CMOS compatible con el Z80. Los chips CMOS (*Complementary Metal Oxide Semiconductor*: semiconductor de óxido metálico complementario) requieren considerablemente menos potencia que los chips de CPU estándares, y este hecho, junto con el empleo del PX-8 de una pantalla LCD de poca potencia, permite que la unidad trabaje enteramente a pilas. Se suministran dos unidades de pilas: una para el consumo principal de potencia y otra como apoyo. Antes de que se pueda utilizar el ordenador se debe cargar la pila, de modo que hay una espera de ocho horas desde que se instala por primera vez la máquina hasta que se halla en condiciones reales de funcionamiento. La unidad principal es recargable y proporciona hasta 15 horas de operación continua antes de que sea necesario volver a cargarla. Según Epson, las expectativas de vida de esta unidad alcanzan a tres o cuatro años.

Una vez que el PX-8 está listo para funcionar, se debe inicializar el sistema operativo. Los pasos necesarios para hacerlo se explican con todo detalle en el manual; éstos implican entrar el día, la fecha y la hora y efectuar algunas tareas "domésticas". Una de éstas es formatear el disco "de RAM". El PX-8 tiene la capacidad de apartar una porción de RAM (que selecciona el usuario entre 9, que es el valor por defecto, y 24 Kbytes) para su empleo como un dispositivo de almacenamiento "en disco". El sistema operativo trata esta zona de la memoria exactamente de la misma forma como lo haría con una unidad de disco externa. Antes de usarlo, el disco de RAM se debe formatear y se debe especificar la cantidad de RAM que utiliza. Asimismo, Epson proporciona una unidad accesoria de disco de RAM, que contiene 120 Kbytes adicionales.

Después de atendidos estos detalles, el PX-8 carga desde la ROM el sistema operativo CP/M y visualiza en la pantalla LCD un directorio de software contenido en ROM y utilidades CP/M en forma de menú. Se puede utilizar software tanto en cassette como en disco o ROM. El software en ROM está retenido en chips EPROM que se colocan en un conector situado debajo de la máquina. El software que se suministra con el PX-8 (Portable Wordstar, Portable Calc y Portable Scheduler) viene en formato de "cápsula", así como el intérprete de BASIC. Para seleccionar una aplicación determinada, se utilizan las teclas del cursor para indicar la opción deseada y luego se pulsa la tecla Return. El programa escogido se carga desde ROM (direccionada por el PX-8 como unidad A y unidad B) en RAM (direccionada como unidad A).

La pantalla LCD proporciona una visualización de 8 líneas de 80 caracteres, con una resolución para gráficos de 480 por 64 pixels. El principal inconveniente de este tipo de pantalla (y, en realidad, el único inconveniente serio de esta excelente máquina) es la lentitud de la visualización. Los caracteres aparecen bastante rápido cuando se los va digitando, pero cualquier borrado (en especial los que implican palabras o frases enteras) es lento.

El software que se suministra con el PX-8 es bastante amplio. Además del procesador de textos, la hoja electrónica y la base de datos ya mencionados, Epson proporciona un programa de telecomunicaciones para utilizar con un modem, y un programa que permite transferir archivos desde el PX-8 a máquinas más grandes, como el QX10 de Epson. Y, puesto que el PX-8 es una máquina CP/M, también puede utilizar el software CP/M ya existente.

El BASIC PX-8 es el Microsoft mejorado por Epson; incluye numeración y renumeración AUTOmática de líneas, un editor completo de pantalla, instrucciones para gráficos y sonido, sentencias que soportan comunicaciones a través de la interface RS232 incorporada, e instrucciones que permiten utilizar la grabadora de microcassette como si se tratara de una unidad de disco (para almacenamiento en acceso directo).

En resumidas cuentas, el Epson PX-8 es un ordenador excelente, ideal, sobre todo, para ejecutivos y periodistas.



Altavoz
Provee la facilidad para ampliar la salida de sonido mediante la conexión con un altavoz externo

Conector A/D
Conecta a instrumentos eléctricos externos

Conector para lector de código de barras
La conexión de un lector de código de barras adecuado permite utilizar el PX-8 para control de precios y stocks

Sub-CPU 7508
Convierte en señales digitales los voltajes variables recibidos en el conector A/D

RAM
El PX-8 contiene los 64 K de RAM necesarios para ejecutar CP/M. El apoyo de la batería asegura la preservación del contenido de la RAM cuando la máquina se apaga



CPU
Versión CMOS del familiar procesador Z80

Puerta para impresora
Conecta al ordenador con una impresora en serie externa

Conector para comunicaciones
Permite que el PX-8 se comunique con otros ordenadores, ya sea directamente o bien a través de un modem y la red telefónica

Puerta para interface en paralelo
Conecta el "disco" de RAM opcional de 120 K

CPU esclava
Permite que el PX-8 se comunique con el altavoz interno del ordenador y periféricos externos tales como una impresora, una unidad de disco o un aparato de cassette

Cápsulas ROM
Permiten el uso de paquetes ROM. Aquí está instalado el Wordstar, sustituible por el Calc

ROM de utilidades CP/M
El PX-8 puede emplear la amplia gama existente de software CP/M

Chris Stevens

EPSON PX-8

DIMENSIONES
297×216×48 mm

CPU
CPU CMOS compatible con Z80, 2,4 MHz

MEMORIA
64 K de RAM, 32 K de ROM más 6 K de RAM de video

PANTALLA
Texto: 8 líneas de 80 caracteres
Gráficos: 480×64 pixels

INTERFACES
RS232, en serie, lector de código de barras, entrada analógica

LENGUAJES DISPONIBLES
BASIC Microsoft perfeccionado operando bajo CP/M

TECLADO
Setenta y dos teclas tipo máquina de escribir, formato QWERTY, incluyendo teclas para control del cursor y cinco teclas de función programables. Hay 12 teclas que se pueden utilizar como un teclado numérico

DOCUMENTACION
Dos grandes volúmenes encuadernados con anillas, un manual de operatoria y guía de referencia del BASIC. Los dos son exhaustivos y están bien escritos

VENTAJAS
Pantalla LCD ancha (80 caracteres) que permite seguir fácilmente lo que está sucediendo en la pantalla; el software basado en ROM simplifica la tarea de cargar los programas en la memoria; fácilmente ampliable

DESVENTAJAS
La pantalla LCD muestra sólo 8 líneas y es de manipulación lenta. Aun con una excelente documentación, el CP/M no es un sistema operativo excesivamente sencillo, en especial para el principiante

Control dual

En este capítulo escribiremos el software para controlar un coche Lego conmutando la corriente desde la caja de salida

Si utilizamos dos motores de igual potencia para activar un vehículo, podemos conseguir un control por ordenador en todas las direcciones de movimiento combinando los movimientos hacia adelante y hacia atrás de cada motor. Esto nos permite hacer girar el vehículo además de moverlo hacia adelante o hacia atrás. Existen, en realidad, dos procedimientos para hacer girar un vehículo de motores gemelos; el primero de ellos es sencillamente detener un motor mientras se hace girar el otro. Esto hará que el vehículo gire en un arco, pivotando alrededor de la(s) rueda(s) estacionaria(s). El segundo método consiste en hacer girar un motor hacia atrás mientras el otro gira hacia adelante, mejorando la maniobrabilidad del vehículo puesto que, al girar, éste pivotará alrededor de su eje central.

Podemos controlar cada motor bidireccionalmente mediante la utilización de las cuatro salidas rojas de la caja de salida de poco voltaje (construida en la p. 1054) y conectando el motor derecho a los terminales 0 y 1, y el motor izquierdo a los terminales 2 y 3 (positivos y negativos, respectivamente).

Cada motor se conecta a través de un par de terminales positivos de salida adyacentes, de modo que podemos obtener un control direccional independiente de cada motor. Colocando ahora el número apropiado en el registro de datos, podemos hacer que el vehículo se desplace hacia atrás o hacia adelante, o que gire hacia la izquierda o la derecha.

Uno, mejor que dos
Habiendo descubierto cómo accionar el vehículo Lego hacia adelante y hacia atrás, ahora podemos unir dos de ellos para hacer un *buggy*. Los motores se pueden conmutar individualmente, con lo que el nuevo vehículo es muchísimo más maniobrable que el anterior

Ian McKinnell

El motor del lado derecho (D) irá hacia adelante si la línea 0 está alta y la línea 1 está baja, e irá hacia atrás si la línea 1 está alta y la línea 0 baja. Del mismo modo, el motor del lado izquierdo (I) irá hacia adelante si la línea 2 se pone alta y la línea 3 baja, y hacia atrás en caso contrario. Mediante la combinación de estos movimientos podemos controlar el de todo el vehículo:

Movimiento del vehículo	Motor I	Motor D	Patrón de bits	N.° en el REGDAT
APAGADO	APAGADO	APAGADO	0000	0
ADELANTE	ADELANTE	ADELANTE	0101	5
ATRAS	ATRAS	ATRAS	1010	10
PIVOTE IZQ.	ADELANTE	ATRAS	0110	6
PIVOTE DER.	ATRAS	ADELANTE	1001	9
ARCO IZQ.	ADELANTE	APAGADO	0100	4
ARCO DER.	APAGADO	ADELANTE	0001	1

El siguiente programa nos permite controlar el vehículo desde el teclado utilizando "T" para marcha adelante, "B" para marcha atrás, "F" para girar a la izquierda, y "H" para girar a la derecha. Si no se pulsa ninguna tecla, el vehículo se detiene.

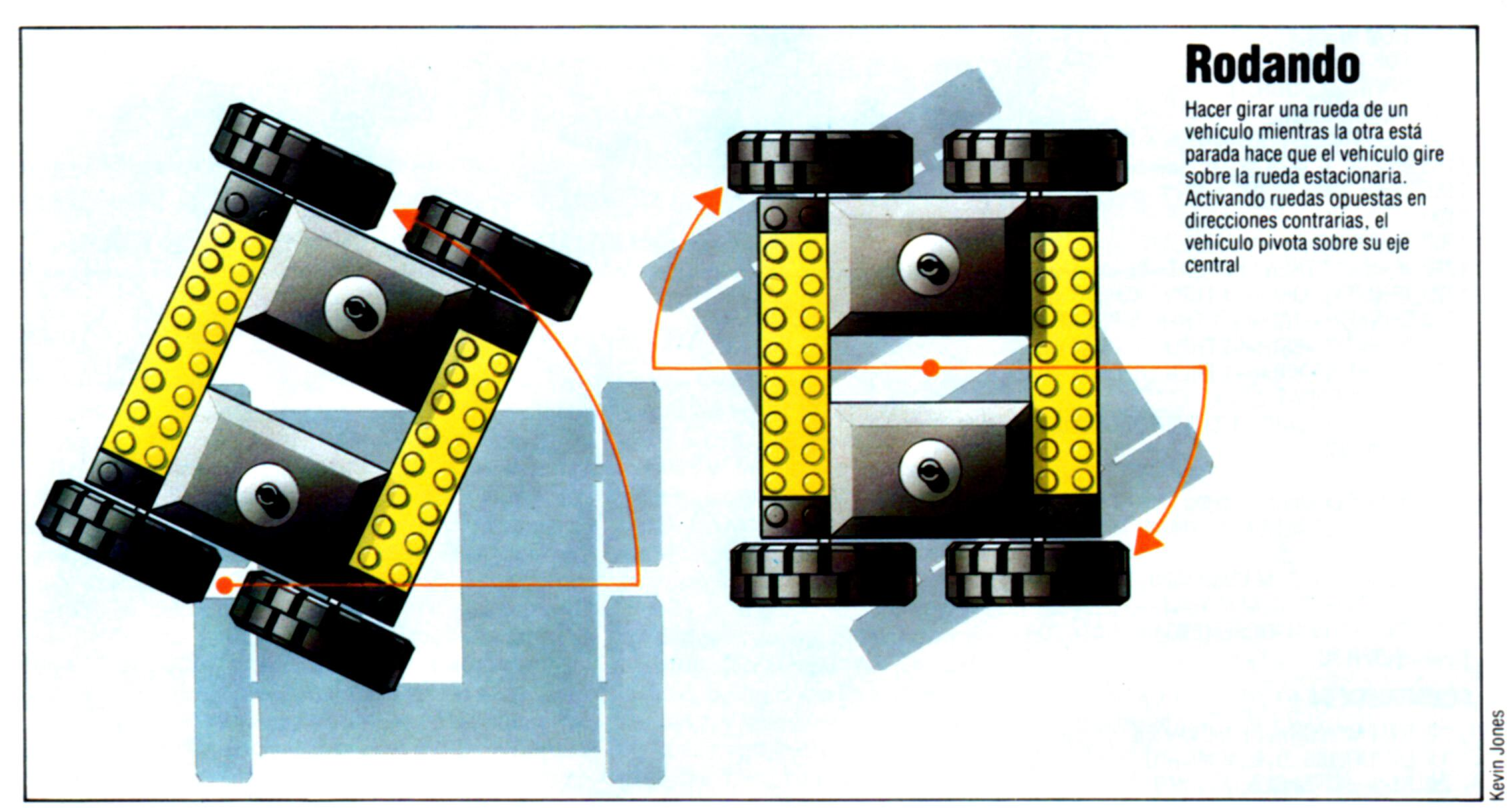
BBC MICRO

```
10 REM MOTORES GEMELOS BBC
20 RDD=&FE62:REGDAT=&FE60
30 ?RDD=255
40 REPEAT
50 A$=INKEY$(10)
60 PROCleer—teclado
70 UNTIL A$="X"
80 ?REGDAT=0
90 END
1000 DEF PROCleer—teclado
1010 IF A$="" THEN ?REGDAT=0
1020 IF INKEY(-36)=-1 THEN ?REGDAT=5
1030 IF INKEY(-101)=-1 THEN ?REGDAT=10
1040 IF INKEY(-68)=-1 THEN ?REGDAT=6
1050 IF INKEY(-85)=-1 THEN ?REGDAT=9
1060 ENDPROC
```

COMMODORE 64

```
10 REM MOTORES GEMELOS CBM 64
20 RDD=56579:REGDAT=56577
25 POKE650,128:REM REPETIR MODO TECLA
30 POKE RDD,255
40 GETA$
50 GOSUB1000:GOTO70
60 POKEREGDAT,0
70 IF A$ <> "X" THEN FOR I=1TO100:NEXT:GOTO40
80 POKE REGDAT,0
90 END
1000 REM COMPROBAR ENTRADA S/R
1005 IF A$="" THEN POKE REGDAT,0
1010 IF A$="T" THEN POKE REGDAT,5
1020 IF A$="B" THEN POKE REGDAT,10
1030 IF A$="F" THEN POKE REGDAT,6
1040 IF A$="H" THEN POKE REGDAT,9
1050 RETURN
```





Rodando

Hacer girar una rueda de un vehículo mientras la otra está parada hace que el vehículo gire sobre la rueda estacionaria. Activando ruedas opuestas en direcciones contrarias, el vehículo pivota sobre su eje central

Kevin Jones

En ambas versiones, el vehículo sólo se moverá mientras se mantenga pulsada una tecla. Apenas se suelte ésta, los motores se pararán al colocarse un cero en el registro de datos. En los dos casos la salida del programa se realiza pulsando "X".

En la versión del programa para el BBC, el procedimiento LEER-TECLADO nos permite verificar el teclado directamente, en vez de leer el buffer del teclado, mediante la utilización de INKEY. Ello nos proporciona una mejor respuesta en el control del vehículo. La versión para el Commodore 64 en primer lugar activa la repetición automática del teclado, de modo que, si se mantiene pulsada una tecla, se seguirán enviando caracteres al buffer del teclado para ser leídos por la instrucción GET. Lamentablemente, no existe ninguna forma de leer el teclado directamente y, por consiguiente, un control exacto es más difícil que en el BBC Micro. La sensibilidad se puede mejorar limpiando el buffer del teclado justo antes de leerlo. Insertando la siguiente línea en la versión del programa para el Commodore 64 se conseguirá esto.

```
35 GET J$:IF J$ <> "" THEN35
```

Además, el GOTO al final de la línea 70 se debe cambiar por GOTO35.

La velocidad a la cual se repite una tecla cuando se la mantiene pulsada puede plantear un problema en las dos versiones de este programa. Si el bucle del programa principal se ejecuta más rápidamente que el tiempo de repetición de una tecla, entonces cuando la rutina tenga que volver a comprobar si se ha pulsado alguna, pensará que no se está pulsando ninguna. Ello provocará un rápido encendido y apagado del motor puesto que la salida alterna rápidamente entre los valores para la dirección elegida y cero. En ambas versiones del programa se ha solventado este problema añadiendo código para aumentar el tiempo de ejecución del bucle del programa principal. En la versión para el BBC, la utilización de INKEY$(10) hace que el ordenador quede "colgado" durante 10 centésimas de segundo, a la espera de una entrada, antes de continuar. En la versión para el Commodore 64 se ha agregado un pequeño bucle de retardo en la línea 70. Los valores de este retardo se hallaron mediante un proceso de ensayo y error, y dependen del lapso necesario para ejecutar una pasada de la rutina. Cuando escriba sus propios programas, quizá se encuentre con que el tiempo de ejecución de la rutina excede a la velocidad de repetición de tecla; de no ser así, inserte en su código un pequeño retardo.

Ahora que hemos obtenido control sobre los movimientos de nuestro vehículo, es interesante diseñar un programa que "memorice" una secuencia de movimientos y los reproduzca. Para hacerlo, podemos valernos de una matriz de dos dimensiones que grabe la dirección y el tiempo consumido en cada una de las maniobras efectuadas. La primera parte de un programa de esta clase será la misma que las que ya hemos dado, pero la segunda reproducirá los datos almacenados. Los datos se almacenarán en una matriz DR(), donde DR(C,1) almacena la dirección y DR(C,2) el tiempo que consume cada movimiento. Cada vez que se selecciona una nueva dirección se utiliza un nuevo elemento de la matriz. Esta condición se indica mediante un cambio en el contenido del registro de datos. Se emplea un contador, C, para indexar la matriz.

BBC MICRO

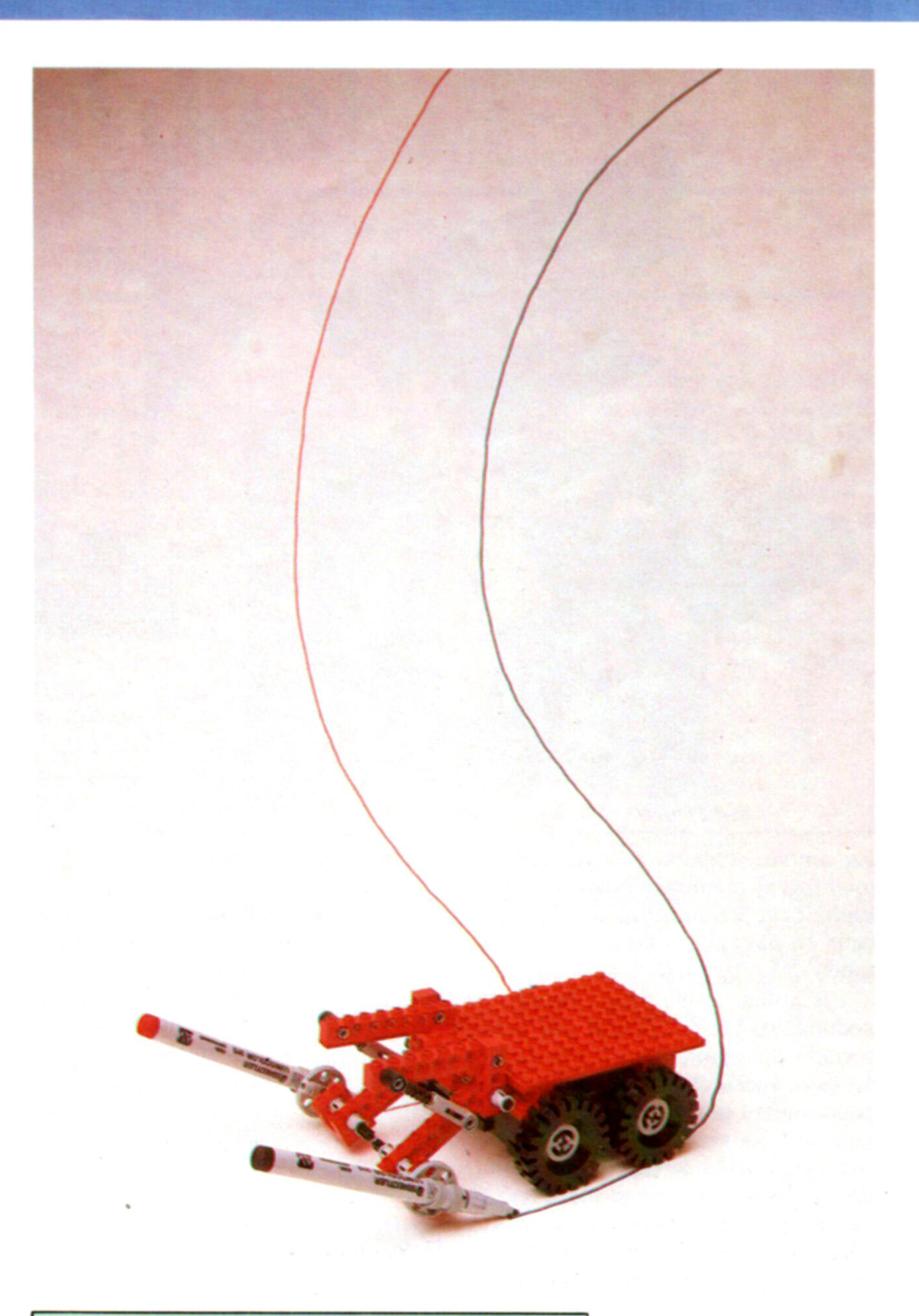
```
1000 REM MEMORIA DE MOVIMIENTOS BBC
1010 RDD=&FE62:REGDAT=&FE60
1020 DIM DR(100,2)
1030 ?RDD=255:C=1:REM INICIALIZAR CONTADOR
1040 REPEAT
1050 A$=INKEY$(10)
1060 PROCleer—teclado
1070 UNTILA$="X"
1080 ?REGDAT=0
1090 DR(C−1,2)=TIEMPO
1100 REPEAT A$=GET$
1110 UNTIL A$="C"
```

```
1120 REM REPRODUCIR DATOS
1130 FOR I=1TOC
1140 ?REGDAT=DR(I,1)
1150 TIEMPO=0
1160 REPEAT UNTIL TIEMPO > =DR(I,2)
1170 NEXT I
1180 END
1190 :
1200 DEF PROCleer—teclado
1210 IF A$="" THEN ?REGDAT=0
1220 IF INKEY(-36)=-1 THEN ?REGDAT=5
1230 IF INKEY(-101)=-1 THEN ?REGDAT=10
1240 IF INKEY(-68)=-1 THEN ?REGDAT=6
1250 IF INKEY(-85)=-1 THEN ?REGDAT=9
1260 PT=?REGDAT
1270 IF PT <> DR(C-1,1)THEN PROCsumar—datos
1280 ENDPROC
1290 :
1300 DEF PROCsumar—datos
1310 DR(C-1,2)=TIEMPO:REM ALMACENAR ULTIMO
     TIEMPO
1320 TIEMPO=0:REM COMENZAR NUEVO TIEMPO
1330 DR(C,1)=PT:REM ALMACENAR ESTADO PUERTA
1340 C=C+1:REM INCREMENTAR CONTADOR
1350 ENDPROC
```

COMMODORE 64

```
  10 REM MEMORIA DE MOVIMIENTOS CBM 64
  15 DIMDR(100,2): REM MATRIZ DE DIRECCION
  20 RDD=56579:REGDAT=56577
  25 POKE650,128:REM ESTABLECER MODO REPETICION
     TECLA
  30 POKERDD,255:REM TODAS SALIDA
  35 C=1:REM INICIALIZAR CONTADOR
  40 GETA$
  50 GOSUB1000:REM COMPROBAR ENTRADA
  70 IF A$ <> "X" THEN FOR I=1TO200:NEXT:GOTO40
  80 POKE REGDAT,0:REM APAGADO
  85 DR(C-1,2)=TI-T:REM ENTRAR ULTIMO TIEMPO
  90 STOP:REM PULSAR "CONT" PARA CONTINUAR
  95 REM REPRODUCIR DATOS
 100 FORI=1TOC
 110 POKEREGDAT,DR(I,1)
 120 T=TI
 130 IF(TI-T) < DR(I,2)THEN130
 140 NEXT
 150 END
 999 :
1000 REM COMPROBAR ENTRADA S/R
1005 IFA$=""THEN POKEREGDAT,0
1010 IFA$="T"THEN POKEREGDAT,5
1020 IFA$="B"THEN POKEREGDAT,10
1030 IFA$="F"THEN POKEREGDAT,6
1040 IFA$="H"THEN POKEREGDAT,9
1045 PT=PEEK(REGDAT)
1050 IF PT <> DR(C-1,1)THENGOSUB1500
1498 RETURN
1499 :
1500 REM AÑADIR DATOS A MATRIZ
1510 DR(C-1,2)=TI-T:REM AÑADIR ULTIMO TIEMPO
1520 T=TI:REM TOMAR NUEVO TIEMPO
1530 DR(C,1)=PT:REM ENTRAR CONTENIDO ACTUAL
     PUERTA
1540 C=C+1:REM INCREMENTAR CONTADOR
1999 RETURN
```

Este programa le permite al usuario desplazar el vehículo controlándolo desde el teclado. Dado que cada movimiento se graba como una dirección y un intervalo de tiempo, cualquier error introducido en el cronometraje de cada movimiento producirá errores en la reproducción. Estamos penetrando en el área de la informática en tiempo real, en donde la estructura del programa y el tiempo de ejecución se pueden convertir en importantes factores.



Movimientos con memoria
Es bastante sencillo escribir un programa para controlar un *buggy* que acepte direcciones desde el teclado y mueva el coche en consecuencia. No es mucho más difícil ampliar el programa de modo que almacene las instrucciones del operador y se las reproduzca después al vehículo, produciendo, por consiguiente (en teoría), el patrón de movimiento anterior. Comparando el original con la supuesta réplica se obtiene una medida de los problemas de software que surgen al tratar con el mundo real: el ordenador trabaja con números y tiempos exactos en un modelo simplificado de un universo perfecto, sin dejar lugar para la inercia, las pérdidas por fricción, las superficies irregulares ni la ingeniería de baja tolerancia. A la luz de esta experiencia, el rendimiento de las tortugas guiadas por LOGO es notable

Ejercicios

Ahora que podemos controlar el movimiento de un vehículo en todas las direcciones, surgen muchas posibilidades para realizar breves ejercicios de programación. Probablemente se le ocurrirán muchas ideas; aquí le proporcionamos algunas:

1) Intente calibrar su vehículo. ¿Durante cuánto tiempo ha de estar el número correspondiente en el registro de datos para hacer que el vehículo se desplace un metro hacia adelante o hacia atrás, o efectúe un giro de 90°?

2) Diseñe un recorrido con obstáculos para su vehículo y, utilizando los programas ofrecidos como base, escriba uno que le permita "enseñarle" a realizar el recorrido bajo el control del teclado. Una vez que lo haya guiado a través de su desplazamiento, el programa deberá invertirse, guiando el vehículo de vuelta hasta su punto de partida.

3) Conecte a la caja buffer cuatro interruptores que le permitan controlar el vehículo externamente desde la puerta para el usuario.

Fuerzas invasoras

Esta vez nos referiremos a la versión original de "Space invaders", popular y pionero juego recreativo de Atarisoft

Casi todos los ordenadores personales que se comercializan actualmente disponen de una versión del *Space invaders* (Invasores del espacio). El juego ha llegado a hacerse tan conocido que a menudo se utiliza como término genérico, hasta el punto de que se suele decir que alguien que esté jugando a cualquier juego recreativo está "jugando a los invasores del espacio".

En 1978, cuando se lanzó este juego, produjo rápidamente una fiebre de proporciones casi epidémicas. Los padres comenzaron a preocuparse por el hecho de que los niños invirtieran todo su tiempo y su dinero deambulando por atestadas salas recreativas. Lo que los celosos guardianes no comprendían era que estos niños en realidad estaban investigando el futuro.

Se puede decir que *Space invaders* cambió la forma en que la sociedad veía a los ordenadores. Antes de que apareciera el juego explotando las capacidades gráficas del microprocesador, los ordenadores se consideraban poco dignos de confianza, siendo el ejemplo clásico el paranoico HAL de la película *2001: una odisea del espacio*.

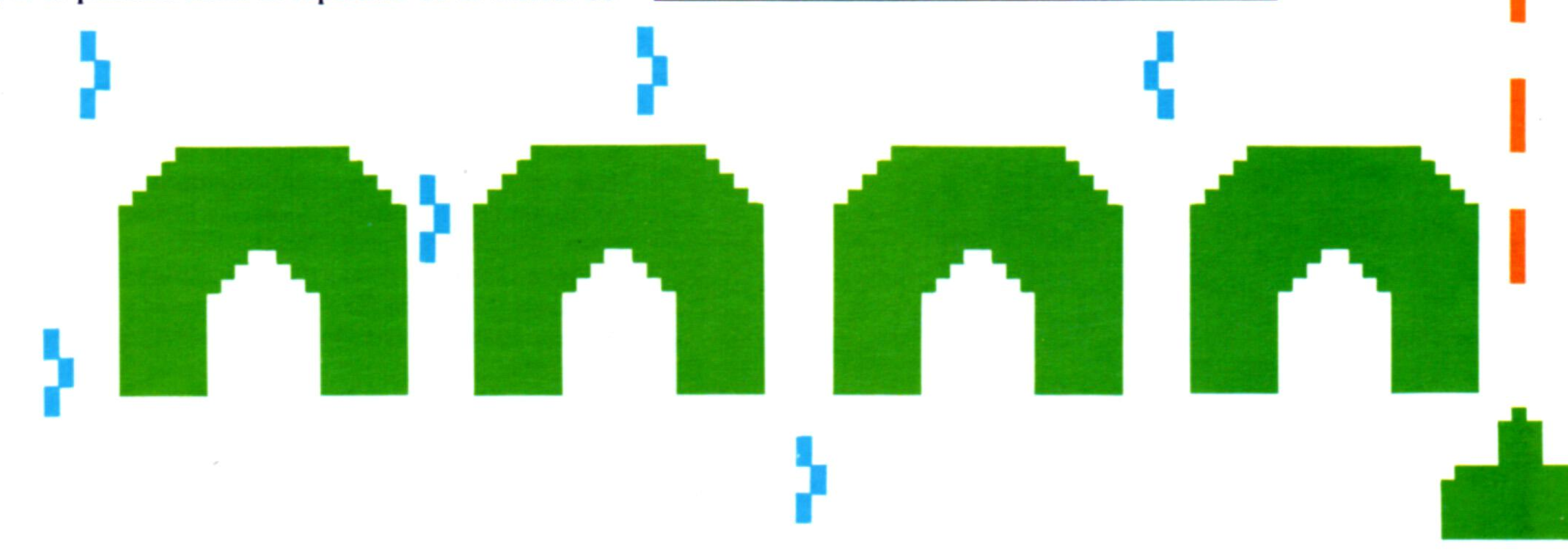
Space invaders fue el precursor de todos los juegos por ordenador del género de "marcianitos". Desde entonces se han producido literalmente centenares de juegos en los que un héroe o una heroína han tenido que enfrentarse a hordas de deleznables seres atacantes sólo con la velocidad de un disparador (y tres vidas) como ayuda.

Es una verdad indiscutible que actualmente *Space invaders* está evidenciando su edad. Para los estándares de hoy en día, el juego es muy simple; pero, a pesar de ello, de todo el software que se ha producido ninguno ha atrapado en tal medida la imaginación del público. El jugador controla una base de láser móvil, que se utiliza para disparar contra formaciones masivas de extraterrestres invasores que van descendiendo amenazadoramente por la pantalla hacia la superficie de la Tierra. Se pierde una "vida" si la base de láser es alcanzada por el fuego de los extraterrestres o si éstos llegan hasta la parte inferior de la pantalla.

Existen varias diferencias entre el juego original y las versiones existentes para ordenadores personales. En vez de aparecer de la nada, ahora las hordas invasoras emergen de un gran cohete situado a la izquierda de la pantalla. Los invasores propiamente dichos son de colores más brillantes y los sprites que los representan son más complejos. Las barreras defensivas, tras las cuales se podía ocultar la base de láser en la versión recreativa, ahora ya no están, y los invasores han de recorrer una distancia más corta para alcanzar la parte inferior de la pantalla. Pero hay un factor que se ha mantenido constante: el amenazador sonido de "latidos de corazón" que acompaña el descenso de los extraterrestres. Éste se vuelve más insistente a medida que los invasores se van acercando y sirve para generar una alta dosis de adrenalina, que probablemente es la principal razón del colosal éxito del juego. Otra característica que comparten las versiones recreativas y para ordenadores personales es el premio "misterio" que se concede cuando el jugador consigue darle a alguno de los platillos volantes que ocasionalmente atraviesan la pantalla de izquierda a derecha.

Space invaders ha conseguido mantener su atractivo durante seis años. A pesar de la existencia de software considerablemente más sofisticado, continúa siendo un juego emocionante y sumamente divertido; es realmente un "clásico del software".

Space invaders: Para todos los ordenadores Atari
Editado y distribuido por: Atari, AUDELEC, Compás de la Victoria, 3, 29012 Málaga, España
Autores: Atari
Palanca de mando: Necesaria
Formato: Cartucho



Liz Dixon

Punteros útiles

Ahora introduciremos mejoras en los programas de definición de caracteres, centrándonos en su almacenamiento y recuperación

Ahora que nuestros programas para definir caracteres están listos y en funcionamiento (véanse pp. 1052 y 1068), vale la pena dedicar un poco de tiempo a comparar las tres versiones y, también, a mejorarlas. La versión para el Commodore se escribió primero, porque es la más difícil de programar de las tres. Esta versión se tradujo luego, virtualmente línea por línea, para las otras dos máquinas. En parte debido a esta traducción y a que el espacio era limitado, el formateado de la pantalla es rudimentario, y no se utiliza ni color, ni sonido, ni gráficos en alta resolución. Es evidente que, en consecuencia, se pueden introducir mejoras en todos estos sentidos, pero no nos ocuparemos de éstas aquí.

Dejando aparte las cuestiones relativas a la eficacia de la programación (no realmente esencial en este programa, dado que no hay tareas que dependan de la velocidad), nos concentraremos en la interface para el usuario: instrucciones, ayudas, teclas de instrucciones y facilidades.

En el programa no hay instrucciones, básicamente porque el listado debía caber en una sola página del curso. Al principio de la ejecución se podría imprimir en la pantalla una página de instrucciones, y en la visualización de la pantalla principal probablemente hay sitio para incluir algunos recordatorios abreviados: una visualización del movimiento del cursor, quizá, o resúmenes de una palabra de las teclas de instrucciones. Ello eliminaría en gran medida la necesidad de una página de ayuda.

La elección de las teclas de instrucciones se podría mejorar. En el BBC Micro y el Spectrum el cursor se mueve por la ventana mediante las teclas normales de control del cursor, mientras que en el Commodore se utilizan las teclas de función sin SHIFT. Esto es mucho más adecuado para la programación de la versión del Commodore, dado que los códigos ASCII de las ocho teclas de función son consecutivos entre 133 y 140, pero el trazado de las teclas no es exactamente ergonómico y no se repiten, a diferencia de las teclas del BBC y el Spectrum. Esto último se puede modificar en el Commodore mediante POKE650,128, pero lo que no se puede es hacer que las teclas de función resulten más fáciles de utilizar, de modo que tal vez desee devolverles el control del cursor a las teclas correspondientes a éste.

Otra posible mejora es la elección de la estrategia para el movimiento del cursor. Tal como está escrita, ésta simplemente desautoriza por ilegal cualquier instrucción que haga que el cursor salga de la ventana. La alternativa consiste en hacer aparecer el cursor por la derecha de la pantalla cuando desaparezca por la izquierda, y viceversa, procediendo de modo similar para el movimiento vertical. Esto es fácil de programar, pero exige más código que el de las comprobaciones simples utilizadas en la subrutina 3500.

Las instrucciones proporcionadas son las mínimas necesarias, y se podrían ampliar. En el Spectrum y en el BBC, guardar y cargar (SAVE y LOAD) juegos de caracteres se podría incluir como instrucciones, y en las tres versiones sería muy útil poder copiar la definición de un carácter en la de otro, de modo que, por ejemplo, CHR$(N) y CHR$(N+1) representaran el mismo carácter. Tal vez se desee una salida impresa del nuevo juego de caracteres, por lo que también se podría agregar una opción para impresora. La sencilla estructura modular del programa hace que estas instrucciones se puedan añadir de forma bastante directa.

El SAVE del Commodore 64

Un problema exclusivo del BASIC Commodore es que la instrucción SAVE parece referirse sólo a la zona entera para programas en BASIC, mientras que los BASIC de las otras dos máquinas permiten que el usuario especifique la zona de memoria que desea guardar. La instrucción LOAD del Commodore, sin embargo, sí permite cargar archivos en cualquier zona deseada, de modo que si pudiéramos solucionar el problema del SAVE podríamos almacenar y recuperar los nuevos juegos de caracteres.

La instrucción SAVE del Commodore está señalada en la zona para programas en BASIC mediante dos punteros de dirección: TXTTAB (en las posiciones 43 y 44) y VARTAB (en las 45 y 46). El primero, TXTTAB, apunta el comienzo de la zona para programas en BASIC (por lo general, a partir de la posición 2048), mientras que VARTAB apunta al comienzo de la zona para variables del BASIC; puesto que ésta comienza donde terminan los programas en BASIC, en realidad VARTAB señala el final de la zona para programas en este lenguaje. Si cambiáramos estos punteros de modo que indicaran el comienzo y el final del nuevo juego de caracteres y generásemos luego una instrucción SAVE, solucionaríamos el problema.

Antes de hacer esto, no obstante, deberíamos reconsiderar la posición del propio juego de caracteres. La subrutina de la línea 61000 (véase p. 1053) copia el juego de caracteres de ROM en un bloque de RAM de dos Kbytes que empieza en 14336, y la






Ian McKinnell

línea 50 coloca el puntero al tope de la memoria al final de este bloque, para evitar que se machaque el BASIC. De esta manera, con el objeto de proteger dos Kbytes estamos recortando dos tercios de la memoria para el usuario. Esto no supone ningún problema para la ejecución del programa generador de caracteres, pero sería una fuente potencial de dificultad si cargáramos el nuevo juego de caracteres en esa dirección para que lo utilice un programa de aplicaciones que necesite más de 12 Kbytes de memoria para el usuario. Lamentablemente, la mecánica del sistema operativo impide que coloquemos el juego de caracteres en un lugar superior de la memoria; si esto fuera posible, podríamos ponerlo arriba, en los dos Kbytes más altos de la memoria para el usuario, o en la zona para programas especiales, desde 49152 en adelante. La solución será colocar el juego de caracteres lo más abajo posible, ¡y desplazar el BASIC por arriba de él! Esto se puede hacer ajustando el contenido de los punteros TXTTAB, pero no desde un programa en BASIC, y debe efectuarse antes de cargar en la memoria el programa generador de caracteres.

La secuencia de acciones es, por consiguiente:
1) Cargar (LOAD) y ejecutar (RUN) el Programa 1. Éste imprime en la pantalla las instrucciones de reubicación necesarias, de modo que puedan ejecutarse en modalidad directa, pulsando Return.
2) Cargar (LOAD) el programa generador de caracteres y efectuar las siguientes modificaciones:

```
61100 CGEN=53248:NCGEN=2048
61500 POKE PO,(PEEK(PO)AND240)OR2
```

y borrar la línea 50.
3) Guardar (SAVE) esta nueva versión.
4) Cargar (LOAD) y ejecutar (RUN) el generador de caracteres exactamente como antes.
5) Cuando termine, cargue y ejecute el Programa 2. Al igual que el Programa 1, imprime en la pantalla instrucciones para que usted las ejecute.
6) El Programa 2 estableció los punteros TXTTAB y VARTAB, de modo que SAVE"nombrearchivo" guarda la zona completa de 2 Kbytes para el juego de caracteres entre 2048 y 4097. En el futuro, para ejecutar el programa generador de caracteres debe repetir esta secuencia, con la excepción del paso 2.

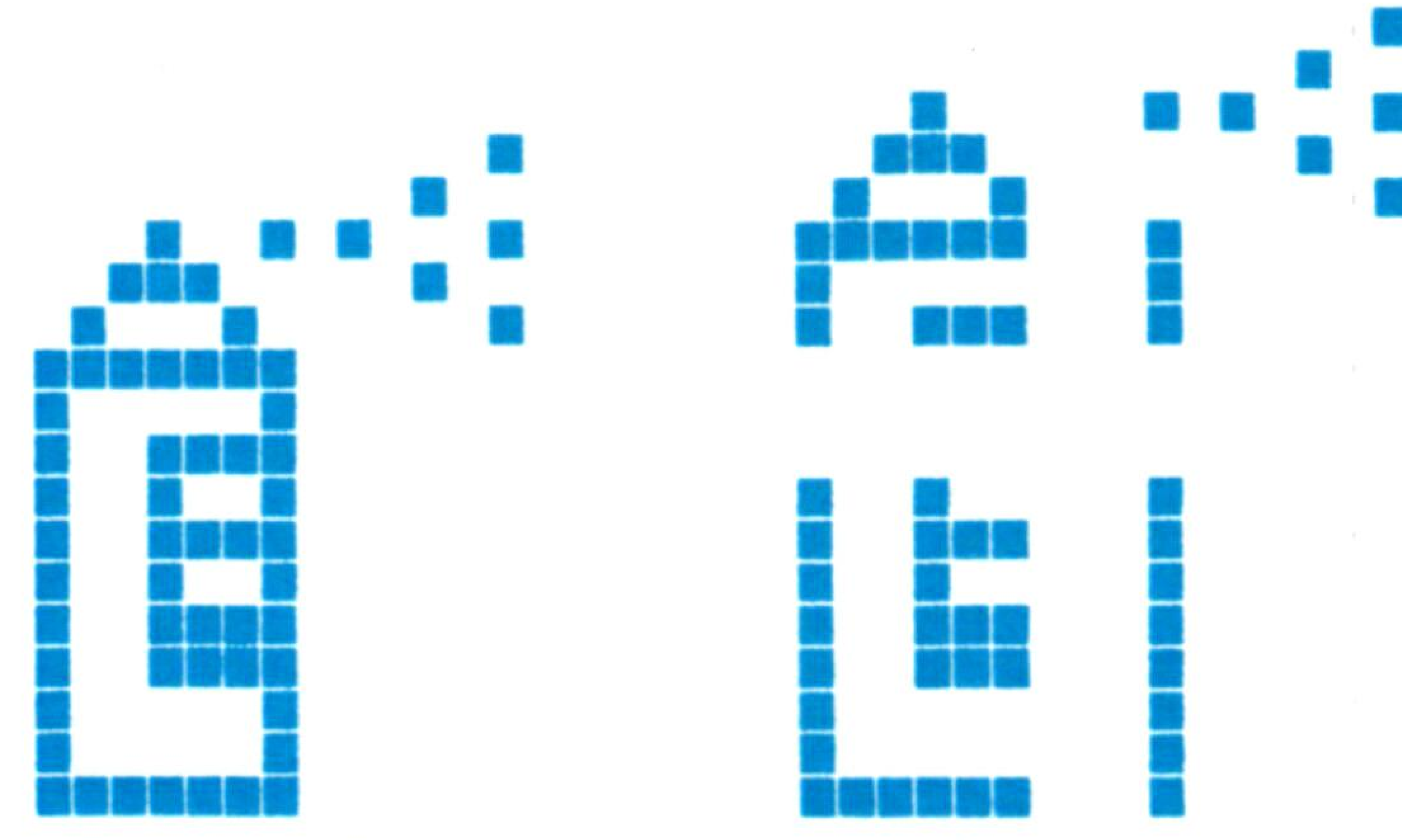
Cuando desee recuperar el juego de caracteres, debe cargar y ejecutar el Programa 1, para desplazar el BASIC hacia arriba de la memoria, y después cargar el juego de caracteres así:

```
LOAD"nombrearchivo",DN,1
```

donde DN (número de dispositivo) es igual a uno para cuando se emplee cassette y es ocho para la unidad de disco. El ",1" al final de la instrucción es conocido como la dirección secundaria, y es la forma de Commodore de enviarles parámetros de instrucciones a los dispositivos periféricos. Por tanto, significa que el archivo se ha de cargar en el lugar de la memoria del cual se guardó, en vez de ser dirigido por el puntero TXTTAB a la zona normal para programas en BASIC. Esto es posible porque cuando se guarda (SAVE) un archivo, el sistema operativo guarda la dirección de comienzo de RAM como el primer dato del archivo. Cuando se emplea la instrucción LOAD sin más, la dirección de comienzo del archivo se ignora, atendiendo en cambio a la dirección que señala TXTTAB.

valor de X=	bits 3,2,1,0	posición a la que se apunta
0	0000	0
2	0010	2048
4	0100	4096
6	0110	6144
8	1000	8192
10	1010	10240
12	1100	12288
14	1110	14336

Tablas de valores
POKE 53272,(PEEK(53272)AND240)OR X obliga al chip de visualización a mirar a la zona de RAM que contiene los caracteres redefinidos. La tabla muestra los valores de X que establecen la dirección de comienzo del bloque de RAM.



Una vez reubicado el BASIC y cargado el nuevo juego de caracteres, debe hacer que el sistema operativo apunte hacia éste; esto se explica en la tabla y se muestra en la nueva versión de la línea 61500.

```
199 REM**********************************
200 REM*            PROGRAMA 1          *
201 REM*      EJECUTE ESTE PROGRAMA     *
202 REM*  LUEGO PULSE RETURN DOS VECES  *
203 REM*     Y EL BASIC PASARA A 4096   *
204 REM**********************************
300 PRINT CHR$(147):PRINT:PRINT
400 PRINT"POKE43,0:POKE44,16:POKE-
    45,3:POKE46,16"
500 PRINT"POKE4096,0:POKE4097,0:POKE
    4098,0:CLR:NEW"
600 PRINT CHR$(19)
```

```
199 REM**********************************
200 REM*            PROGRAMA 2          *
201 REM*      EJECUTE ESTE PROGRAMA     *
202 REM*  LUEGO PULSE RETURN DOS VECES. *
203 REM* SE RESTAURAN LOS PUNT. DE BASIC *
204 REM**********************************
300 PRINT CHR$(147):PRINT:PRINT
400 PRINT"POKE43,0:POKE44,8:POKE-
    45,1:POKE46,16"
500 PRINT"POKE4096,0:POKE4097,0:POKE
    4098,0:CLR"
600 PRINT CHR$(19)
```

Con su pareja

Después del estudio del direccionamiento indexado, en esta ocasión vamos a analizar las operaciones aritméticas y las subrutinas comparativas

La lección anterior trató del direccionamiento indexado en el 6809. En este modo de direccionamiento, la dirección efectiva especificada por DESPL,X se forma como la suma del desplazamiento (que puede ser una constante o el contenido de una posición de memoria) y el valor que contenga el registro índice especificado (en este caso, el X). Vimos que en algunos casos comunes, el desplazamiento puede ser cero, en cuyo caso podemos escribir ,X (también funcionaría 0,X). En algunas circunstancias especiales podemos servirnos de acumuladores A,B o D para expresar el desplazamiento (p. ej., B,X). Analizamos también la manera de hacer más fácil el uso del modo autoincremento y autodecremento, para uno de los empleos más frecuentes del direccionamiento: el de recorrer una tabla de valores. Tales modos incrementan el registro en una o en dos unidades una vez ejecutada la instrucción (,X+ y ,X++), o bien decrementan éste de la misma manera pero antes de ejecutar la instrucción (,−Y y ,−−Y).

Vamos ahora a examinar brevemente la forma como podemos emplear el direccionamiento indexado para realizar sencillas operaciones aritméticas con valores contenidos en los registros índice gracias a la instrucción LEA (*Load Effective Address*: cargar la dirección efectiva). Las diversas instrucciones normales de tipo aritmético sólo funcionan en los acumuladores, no en los restantes registros. Sin duda que podemos transportar el contenido del registro índice al acumulador D, realizar la operación y devolver el resultado al registro, pero éste es un procedimiento lento y engorroso. La instrucción LEA (sólo aplicable a los registros X,Y,S y U) hará los cálculos necesarios de la dirección y cargará el valor efectivo de ésta. Como quiera que es frecuente cargar el contenido de una dirección efectiva, nos encontramos ante una alternativa digna de tener en cuenta.

Veamos un ejemplo. La instrucción

```
LEAX    -1,X
```

calculará la dirección efectiva sumando −1 al valor actual del registro X. Esta dirección se carga en X, decrementando efectivamente el valor previo del registro. Pero la instrucción no sólo sirve para esto. Puede también utilizarse, por ejemplo, para calcular una sola vez una dirección y guardar el resultado, con objeto de evitar repetir el mismo cálculo varias veces.

Se puede igualmente realizar cierto número de operaciones aritméticas sobre el registro X mediante ABX (*Add* B *to* X: sumar B a X), una instrucción que sirve para sumar sin signo valores contenidos en B a los contenidos en X. Pero no resulta tan útil como LEA.

Subrutinas

Una subrutina es un fragmento autosuficiente de código máquina que pueda ser llamado por el programa principal (o por otra subrutina) para realizar una tarea específica. Una vez efectuada ésta, se trasfiere automáticamente el control al programa o rutina que la llamó (*call*) para continuar con la instrucción inmediatamente siguiente a la de la llamada. Hay, al menos, tres razones para emplear subrutinas:

1) Porque nos ahorramos el escribir el mismo fragmento cada vez que lo necesitemos. Es mucho mejor escribirlo una sola vez como subrutina y llamarlo cada vez que sea necesario su uso.
2) Porque podemos hacernos con toda una "subrutinoteca" y utilizar las rutinas en programas diferentes.
3) Porque así se subdivide el programa en trozos más pequeños y más manejables.

Lo que hay que tener más en cuenta en las subrutinas del assembly es que emplean los mismos registros que el programa que las llama. Uno de los errores más comunes en la programación de código máquina consiste en que, una vez almacenado un valor en alguno de los registros, el programa puede llamar a una subrutina que altera dicho valor y el programador no lo tiene en cuenta en el momento de devolver el control. Por ello es esencial conocer y documentar los registros que utiliza la subrutina. Es especialmente importante salvar los contenidos de los registros que van a ser utilizados por la subrutina y restaurarlos una vez que la subrutina haya completado su tarea.

Más adelante nos detendremos a examinar cómo se emplean las pilas tanto para salvar tales datos como para pasar parámetros (valores y direcciones) a la subrutina. Por el momento asumiremos que la subrutina emplea los mismos datos que el programa que la llama (variables globales) y que cualquier otro valor que necesite se encontrará en los registros. La llamada de una subrutina se hace mediante una de estas instrucciones:

- BSR (*Branch to SubRoutine*: bifurcación a subrutina)
- JSR (*Jump to SubRoutine*: salto a subrutina)

La orden BSR provoca una bifurcación relativa, es decir, encuentra la subrutina en un cierto desplazamiento respecto al valor actual del contador de programa. Se suele utilizar esta instrucción para subrutinas escritas formando parte del programa.

La instrucción JSR llama a la subrutina especificando una dirección determinada. Se emplea cuando la subrutina se retiene en la ROM o bien cuando



se dispone de una biblioteca de rutinas situada siempre en el mismo lugar de la memoria, por ejemplo formando parte del sistema operativo de disco.

Cuando el procesador encuentra una instrucción BSR o JSR, el valor en curso del contador de programa es colocado (*push*) en la pila del sistema utilizando el registro S (puntero o índice de la pila). Si esta subrutina que llamamos emplea el registro S para cualquier otro objeto aparte del de llamar a su vez a otra subrutina, nunca deberemos olvidarnos de restituir el valor correcto. El resultado del cálculo de la dirección de la subrutina (para el caso de BSR) se carga en el contador de programa. De esta manera la siguiente instrucción que hay que ejecutar coincidirá con la primera de la subrutina. Asegúrese, por lo tanto, de que la subrutina comienza con una instrucción y no con algún byte de datos.

La subrutina debe concluir con una instrucción RTS (R*e*T*urn from* S*ubroutine*: volver de la subrutina), que tiene por efecto el de extraer (*pull*) de la pila el valor anterior para reponerlo en el contador del programa. La ejecución de éste se reanudará allí donde quedó interrumpida por causa de la llamada a la subrutina.

El programa de ejemplo que damos aquí resulta algo más complicado de lo acostumbrado, pero se puede hacer más manejable si empleamos una subrutina. Se trata de buscar una tabla que contiene series o cadenas (*strings*) de desigual longitud y extraer un valor que se corresponda con una determinada serie. Se han colocado las series de la manera habitual: se inician con un byte que indica la longitud de la serie y seguidamente van los caracteres que la componen para finalizar con una dirección de 16 bytes que corresponde a esa cadena.

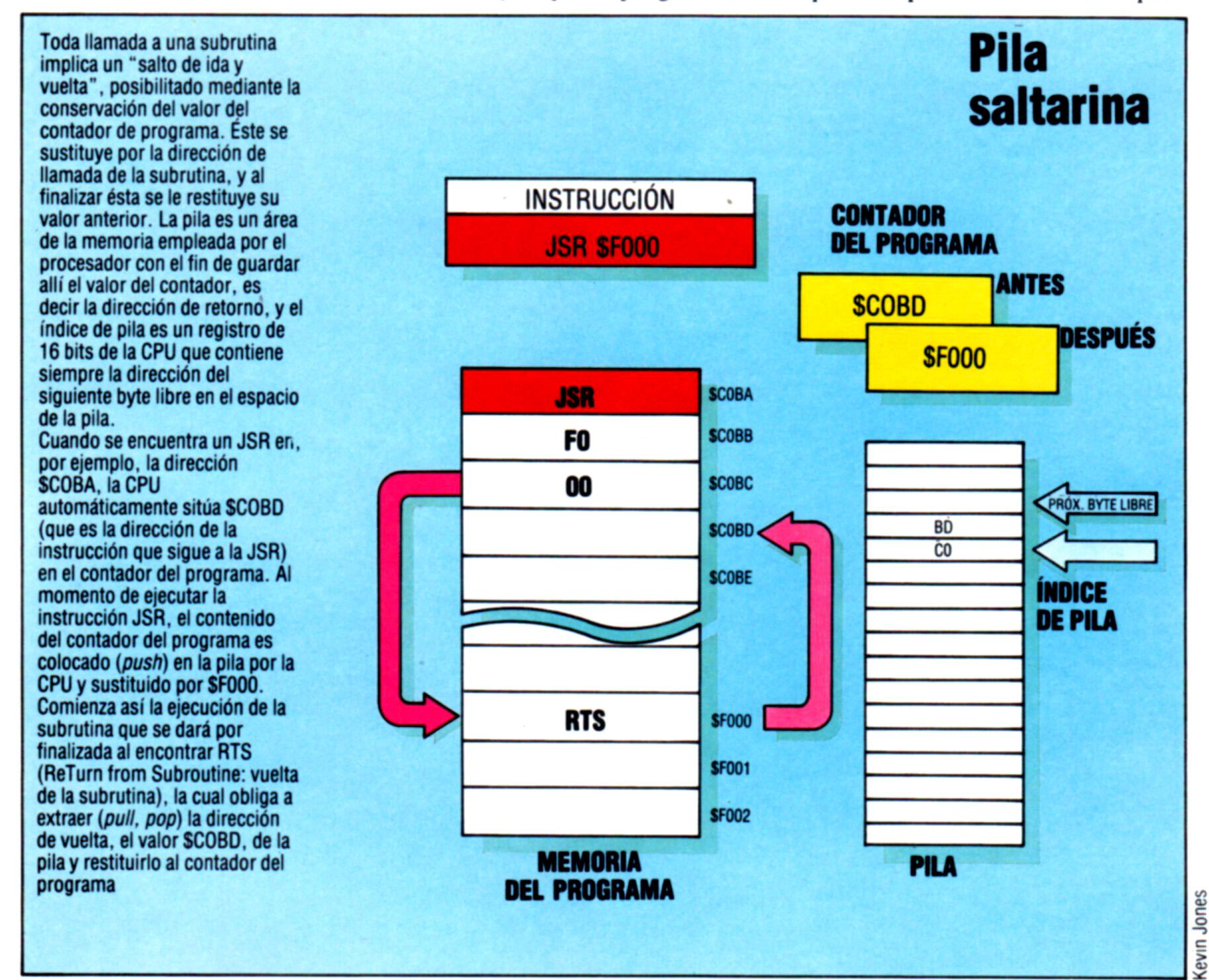
El final de la tabla se señala con una cadena de longitud cero, o sea, se encuentra el cero donde debería encontrarse el byte de longitud. Supondremos que la dirección de comienzo de la tabla se sitúa en $10, y que la dirección de la cadena cuya correspondencia buscamos está en $12. Si se encuentra la pareja en la tabla, la dirección correspondiente se colocará en $14. Pero si no es hallada, se pondrá a cero tanto $12 como $14.

Emparejamiento de series

El emparejamiento de series o cadenas es una tarea que se da en múltiples ocasiones, especialmente cuando se manejan accesos o variables alfanuméricas por medio del intérprete BASIC: cada identificador (o nombre de la variable) debe ser sustituido por las direcciones en las que está almacenado el valor de esa variable.

Es fácil dividir el problema en dos partes: hay que recorrer la tabla hasta que encontremos o bien la cadena que buscamos o bien el final de la tabla. Cada vez que realicemos un movimiento de búsqueda habremos de comparar dos cadenas por si coinciden: la que buscamos y la que se encuentra en la actual posición en la tabla.

Esta comparación de series se presta obviamente a ser realizada mediante una subrutina, porque no sólo habrá de usarse más de una vez en el programa, sino que nos permitirá dividir el pro-



Pila saltarina

Toda llamada a una subrutina implica un "salto de ida y vuelta", posibilitado mediante la conservación del valor del contador de programa. Éste se sustituye por la dirección de llamada de la subrutina, y al finalizar ésta se le restituye su valor anterior. La pila es un área de la memoria empleada por el procesador con el fin de guardar allí el valor del contador, es decir la dirección de retorno, y el índice de pila es un registro de 16 bits de la CPU que contiene siempre la dirección del siguiente byte libre en el espacio de la pila.
Cuando se encuentra un JSR en, por ejemplo, la dirección $COBA, la CPU automáticamente sitúa $COBD (que es la dirección de la instrucción que sigue a la JSR) en el contador del programa. Al momento de ejecutar la instrucción JSR, el contenido del contador del programa es colocado (*push*) en la pila por la CPU y sustituido por $F000. Comienza así la ejecución de la subrutina que se dará por finalizada al encontrar RTS (ReTurn from Subroutine: vuelta de la subrutina), la cual obliga a extraer (*pull, pop*) la dirección de vuelta, el valor $COBD, de la pila y restituirlo al contador del programa

blema en dos útiles secciones. Además de ser una subrutina digna de tenerla disponible para otros programas.

La subrutina necesita dos datos suministrados por el programa que la llama: las direcciones de las dos cadenas a comparar. Puesto que la subrutina tiene que repasar las cadenas byte a byte, será mejor trasladar estos valores a los registros X e Y, donde serán necesarios. Además, la subrutina ha de retornar dos valores, uno que indique si hubo o no coincidencia, y otro que diga la dirección de la pareja encontrada.

Verdadero o falso

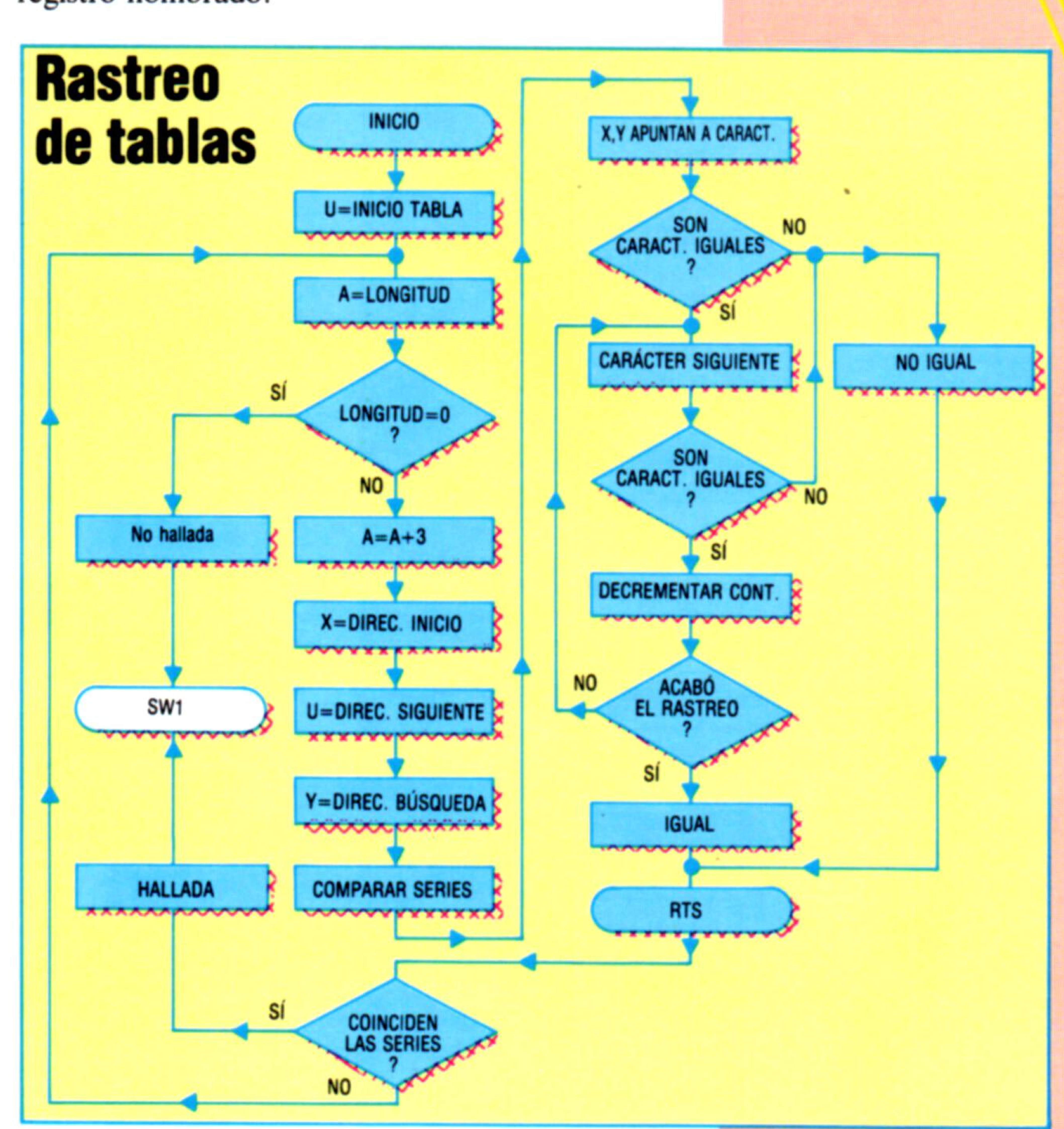
Es posible pasar un parámetro booleano (verdadero o falso) mediante los flags del registro de código de condición, pero es preciso tener puntual conocimiento del efecto de cada instrucción sobre los flags. En nuestro programa los valores que llevaremos a la rutina principal serán o todos ceros ($00), o todos unos ($FF), según que hubo coincidencia o no, respectivamente.

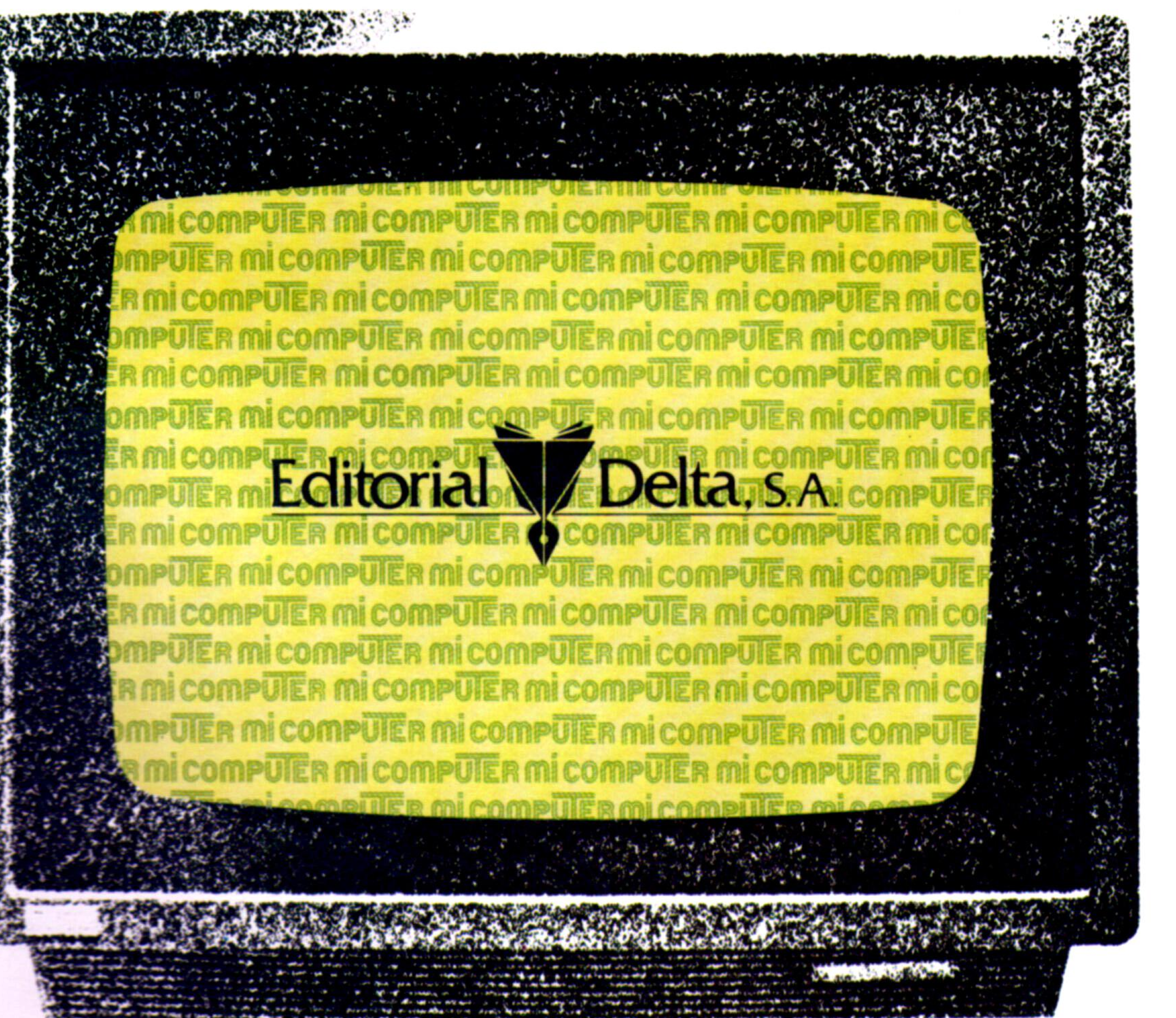
Para generalizar la subrutina, no suministraremos la dirección concreta de la pareja hallada, sino que dejaremos que el registro quede señalando la dirección donde se encuentra la dirección de la pareja hallada. Lo cual tiene una ventaja adicional, y es que el registro X, al recorrer la cadena byte a byte, acabará conteniendo esta información de manera automática.

Una última observación: nuestro programa contiene una nueva instrucción del 6809. Es TST (*TeST*: comprobación), la cual no afecta a registro alguno y se limita a activar los flags según el valor actual del registro nombrado.



```
TABLA   EQU   $10
STRNG   EQU   $12
ADDRS   EQU   $14
        ORG   $1000
        LDU   TABLA
BUCLE1  LDA   ,U
        BEQ   NTFND1
        ADDA  #3
        TFR   U,X
        LEAU  A,U
        LDY   STRNG
        BSR   COMPAR
        TSTA
        BEQ   FOUND1
        BRA   BUCLE1
FOUND1  LDD   ,X
        BRA   FINAL1
NTFND1  LDD   #0
FINAL1  STD   ADDRS
        SWI
COMPAR  LDB   ,X+
        CMPB  ,Y+
        BNE   NOTEQ
BUCLE2  LDA   ,X+
        CMPA  ,Y+
        BNE   NOTEQ
        DECB
        BGT   BUCLE2
        CLRA
        BRA   FINAL2
NOTEQ   LDA   #$FF
FINAL2  RTS
        END
```



Editorial Delta, S.A.
mi computer



10055
9 788485 822836